(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **2002175344 A**

(43) Date of publication of application: **21.06.02**

(51) Int. Cl **G06F 17/50**
**G01R 31/28**
**G06F 11/26**

(21) Application number: **2001274491**

(22) Date of filing: **11.09.01**

(30) Priority: **17.10.00 US 2000 688745**

(71) Applicant: **NEC CORP**

(72) Inventor: **LESLIE J FRENCH**

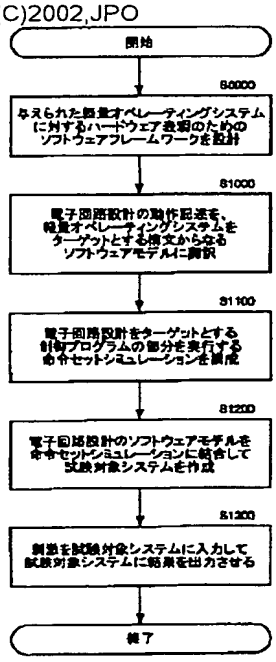
# (54) CO-VALIDATION METHOD BETWEEN ELECTRONIC CIRCUIT AND CONTROL PROGRAM



(57) Abstract:

PROBLEM TO BE SOLVED: To use a light-weight operating system on a message base as a base for co-design (cooperative design) and co-simulation (cooperative simulation) of hardware/software.

SOLUTION: Circuit design is accepted operation description and is mapped to software processes. A VHDL process is translated by using a rule set designed for a specified light weight OS to be used. Hardware processes, mapped to the software mutually communicate state changes through message transfer primitive of the light weight OS, using an interface library. Integrated co- simulation environment of the software (to be written, for example, in C or assembler language) and the hardware (to be written, for example, in VHDL and a subset of C) is obtained by the addition of an instruction set simulator which operates under the same OS is added.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－175344
（P2002－175344A）
(43)公開日　平成14年６月21日(2002.6.21)

| (51)Int.Cl.⁷ | 識別記号 | ＦＩ | | テーマコード（参考） |
|---|---|---|---|---|
| Ｇ０６Ｆ　17/50 | ６６４ | Ｇ０６Ｆ　17/50 | ６６４Ａ | ２Ｇ１３２ |
| | | | ６６４Ｊ | ５Ｂ０４６ |
| Ｇ０１Ｒ　31/28 | | 11/26 | | ５Ｂ０４８ |
| Ｇ０６Ｆ　11/26 | | Ｇ０１Ｒ　31/28 | Ｆ | |

審査請求　有　　請求項の数25　ＯＬ　（全 33 頁）

(21)出願番号　特願2001－274491(P2001－274491)

(22)出願日　平成13年９月11日(2001.9.11)

(31)優先権主張番号　０９／６８８７４５

(32)優先日　平成12年10月17日(2000.10.17)

(33)優先権主張国　米国（ＵＳ）

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

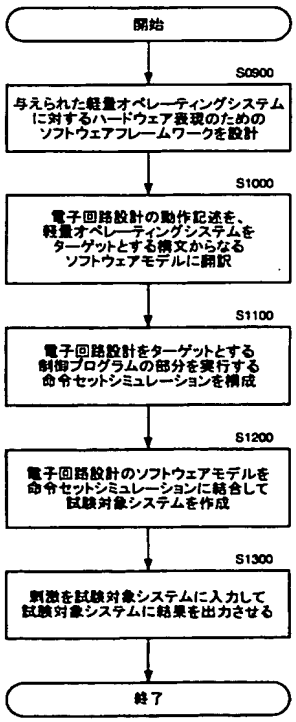
(72)発明者　レスリー・ジェイ・フレンチ
アメリカ合衆国，ニュージャージー　08540　プリンストン，４　インディペンデンス　ウエイ，エヌ・イー・シー・ユー・エス・エー・インク内

(74)代理人　100097157
弁理士　桂木　雄二

Ｆターム(参考)　2G132 AC09
5B046 AA08 BA03 JA05
5B048 DD14

(54)【発明の名称】　電子回路と制御プログラムとのコバリデーション方法

(57)【要約】
【課題】　ハードウェア／ソフトウェア・コデザイン（協調設計）およびコシミュレーション（協調シミュレーション）の基礎として、メッセージベースの軽量オペレーティングシステムを利用する。
【解決手段】　回路設計を動作記述として受け入れ、ソフトウェアプロセスにマッピングする。ＶＨＤＬプロセスは、使用される特定の軽量ＯＳ用に設計された規則セットを用いて翻訳される。ソフトウェアにマッピングされたハードウェアプロセスは、インタフェースライブラリを用いて軽量ＯＳのメッセージ受渡しプリミティブを通じて状態変化を通信しあう。同じＯＳの下で動作する命令セットシミュレータが付加されることにより、ソフトウェア（例えばＣやアセンブラで書かれる）とハードウェア（例えばＶＨＤＬやＣのサブセットで書かれる）の統合コシミュレーション環境が得られる。



1

【特許請求の範囲】

【請求項１】　電子回路と該電子回路をターゲットとする制御プログラムとのコバリデーションを行う方法において、前記電子回路および前記制御プログラムが軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、
前記方法は、
前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文からなるソフトウェアモデルに、前記電子回路の動作シミュレーションを翻訳するステップと、
命令セットシミュレーションが前記電子回路をターゲットとする制御プログラムの一部を実行するように、サイクル精度の命令セットシミュレーションを構成するステップと、
前記電子回路のソフトウェアモデルを前記命令セットシミュレーションに結合して、前記軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成するステップと、
前記試験対象システムに刺激を入力することにより、前記試験対象システムに結果を出力させるステップと、
を有することを特徴とする電子回路と制御プログラムとのコバリデーション方法。

【請求項２】　前記動作シミュレーションを翻訳するステップは、複数の規則を用いて、前記動作シミュレーションを、前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文に変換するステップを有することを特徴とする請求項１記載の方法。

【請求項３】　前記複数の規則のうちの１つは、動作シミュレーション代入を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする内部変数構文に変換することを特徴とする請求項２記載の方法。

【請求項４】　前記複数の規則のうちの１つは、動作シミュレーション信号代入を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする外部変数構文に変換することを特徴とする請求項２記載の方法。

【請求項５】　前記複数の規則のうちの１つは、動作シミュレーションウェイトポイントを、前記軽量コンピュータシステムオペレーティング環境をターゲットとするウェイト構文に変換することを特徴とする請求項２記載の方法。

【請求項６】　前記複数の規則のうちの１つは、動作シミュレーション制御構文を、前記軽量コンピュータシステムオペレーティング環境をターゲットとする制御構文に変換することを特徴とする請求項２記載の方法。

【請求項７】　前記サイクル精度の命令セットシミュレーションを構成するステップは、中央処理装置をターゲットとするシミュレーションを構成するステップを含むことを特徴とする請求項１記載の方法。

【請求項８】　前記命令セットシミュレーションを構成

2

するステップは、グルーロジックを通じてターゲット中央処理装置に接続されたハードウェアデバイスを表すようにアドレス範囲をマッピングするステップを含むことを特徴とする請求項７記載の方法。

【請求項９】　前記命令セットシミュレーションを構成するステップは、ターゲット中央処理装置のキャッシュ動作をエミュレートするステップを含むことを特徴とする請求項７記載の方法。

【請求項１０】　前記命令セットシミュレーションを構成するステップは、ターゲット中央処理装置の命令スケジューリングをエミュレートするステップを含むことを特徴とする請求項７記載の方法。

【請求項１１】　前記軽量コンピュータシステムオペレーティング環境をターゲットとする構文は、前記軽量コンピュータシステムオペレーティング環境をターゲットとするインタフェースライブラリの一部を含むことを特徴とする請求項１記載の方法。

【請求項１２】　電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
前記電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、
前記コンピュータシステムは、
すべてのグローバル信号の状態を管理するソフトウェア手段と、
クロック信号を生成するソフトウェア手段と、
イベントをウェイトするプロセスキューを管理するソフトウェア手段と、
所定のタイミング間隔で少なくとも１つのプロセスを実行するソフトウェア手段と、
中央プロセス制御を行うソフトウェア手段と、
を有することを特徴とするコンピュータシステム。

【請求項１３】　前記グローバル信号に対する状態変化を表示するディスプレイを駆動するソフトウェア手段をさらに有することを特徴とする請求項１２記載のコンピュータシステム。

【請求項１４】　電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
前記電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされ、
前記コンピュータシステムは、
プロセッサと、
前記コンピュータシステムが中央制御プロセスを実行することを可能にするように適応したソフトウェア命令を含むメモリと、を有し、
前記中央制御プロセスは、
複数のグローバル信号の状態を管理する信号メンテナサ

ブプロセスと、
ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスと、
ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、
所定のタイミング間隔を生成するタイマサブプロセスと、
少なくとも前記信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御するコントローラサブプロセスと、を有することを特徴とするコンピュータシステム。

【請求項１５】　前記所定のタイミング間隔は、１ナノ秒以上の時間間隔をシミュレートすることを特徴とする請求項１４記載のコンピュータシステム。

【請求項１６】　前記所定のタイミング間隔は、１ナノ秒未満の時間間隔をシミュレートすることを特徴とする請求項１４記載のコンピュータシステム。

【請求項１７】　前記グローバル信号に対する状態変化を表示するグローバルディスプレイサブプロセスをさらに有することを特徴とする請求項１４記載のコンピュータシステム。

【請求項１８】　マイクロプロセッサによって制御される電子回路と、該マイクロプロセッサをターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムにおいて、
前記電子回路、ターゲットマイクロプロセッサおよび制御プログラムは、所定のコンピュータ言語および所定のインタフェースライブラリを用いてシミュレートされ、
前記コンピュータシステムは、
前記電子回路の機能が所定のインタフェースライブラリの一部を含む所定のコンピュータ言語構文に翻訳されている前記電子回路のソフトウェアモデルと、
前記制御プログラムの一部を実行するときに命令セットシミュレーションが前記ターゲットマイクロプロセッサによって実行される命令をシミュレートするような、前記ターゲットマイクロプロセッサの命令セットシミュレーションと、
前記電子回路のソフトウェアモデルと、前記ターゲットマイクロプロセッサの命令セットシミュレーションを互いに結合する中央制御プロセスと、を有し、
前記中央制御プロセスは、
複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、
前記ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスと、
前記ソフトウェアモデルおよび命令セットシミュレータ



からのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、
所定のタイミング間隔を生成するタイマサブプロセスと、
少なくとも前記信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御する制御サブプロセスと、を有することを特徴とするコンピュータシステム。

【請求項１９】　前記所定のタイミング間隔は、１ナノ秒以上の時間間隔をシミュレートすることを特徴とする請求項１８記載のコンピュータシステム。

【請求項２０】　前記所定のタイミング間隔は、１ナノ秒未満の時間間隔をシミュレートすることを特徴とする請求項１８記載のコンピュータシステム。

【請求項２１】　前記複数のグローバル信号に対する状態変化を表示する信号ディスプレイサブプロセスをさらに有することを特徴とする請求項１８記載のコンピュータシステム。

【請求項２２】　電子回路と、該電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムのための実行可能プログラムにおいて、
前記電子回路および制御プログラムは、所定のコンピュータ言語を用いてシミュレートされ、
前記実行可能プログラムは、
コンピュータ上で実行されるときに、複数のグローバル信号の状態を管理する第１実行可能コード部分と、
コンピュータ上で実行されるときに、複数のクロック信号を生成する第２実行可能コード部分と、
コンピュータ上で実行されるときに、イベントをウェイトするプロセスキューを管理する第３実行可能コード部分と、
コンピュータ上で実行されるときに、所定のタイミング間隔を生成する第４実行可能コード部分と、
コンピュータ上で実行されるときに、少なくとも前記第１、第２、第３および第４実行可能コード部分の実行を制御する第５実行可能コード部分と、
を有することを特徴とする、電子回路と制御プログラムのコバリデーションを行うコンピュータシステムのための実行可能プログラム。

【請求項２３】　電子回路と、該電子回路を制御する制御プログラムとを含むシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法において、
メモリオペレーションの完了を待機することによってキャッシュがロードされるまで前記ターゲットプロセッサがストールされるように、メモリとキャッシュの間の相互作用をデータバス上の信号のシーケンスとしてモデル化するステップと、

前記ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いるとともに、内部バス幅およびタイミングを用いて、命令パイプラインを充填して、適当なクロックサイクル数だけ遅延を行うステップと、

前記ターゲットプロセッサの命令デコードサイクルを実行して、実行のために利用可能な命令を解釈するステップと、

スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定し、スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをストールさせるステップと、

スケジューリングされた命令を、適当な命令パイプラインまたはハードウェアコンポーネントに転送し、各命令ごとにサイクル実行時間を計算するステップと、

非決定性タイミングがあるかどうかを判定するステップと、

計算された実行サイクルの終端で利用可能な結果を出力するステップと、

前記ターゲットプロセッサのエミュレートされた制御レジスタとともに、信号インタフェースを用いて、割込みハンドラが次のサイクルにスケジューリングされるべきかどうかを決定するステップと、

を有することを特徴とする命令セットシミュレーションを導出する方法。

【請求項２４】　前記非決定性タイミングがあるかどうかを判定するステップは、

非決定性タイミングがある場合、各クロックサイクルにおいて終了条件を評価するステップと、

非決定性タイミングがない場合、計算されたタイミングに対するリソース割当てを伝搬させるステップと、

を含むことを特徴とする請求項２３記載の方法。

【請求項２５】　前記スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定するステップは、

スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをストールさせるステップを有することを特徴とする請求項２３記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、軽量のメッセージベースのオペレーティングシステムを用いてハードウェアおよびソフトウェアのコデザイン（co-design、協調設計）およびコバリデーション（co-validation、協調検証）を行う方法および装置に関する。特に、電気回路設計はソフトウェアプロセスにマッピングされ、マッピングされた電気回路要素間の状態変化は、軽量オペレーティングシステムのメッセージ受渡しプリミティブを用いて渡される。軽量オペレーティングシステム下で動作する命令セットシミュレータにより、制御ソフトウェア要素と、マッピングされた電気回路要素とはコシミュレート（co-simulate、協調シミュレート）され、それらの相互作用の評価が可能となる。本発明は、ソフトウェアおよびハードウェアのコシミュレーション（co-simulation、協調シミュレーション）環境を提供するコバリデーション方法、コシミュレーション方法をインプリメント（実装）するコンピュータシステム、コシミュレーション方法をインプリメントするソフトウェア命令を含むコンピュータプログラム製品、ならびに、ハードウェア要素のソフトウェアシミュレーションおよび命令セットシミュレーションをサポートするソフトウェア環境として、実現される。また、本発明は、ターゲット中央処理装置（ＣＰＵ）に対する命令セットシミュレータを開発する方法としても実現される。



【０００２】

【従来の技術】以下の参照事項は、すべて本発明に関連する示した主題についての有用な背景的情報を提供する。

【０００３】ＰＴＯＬＥＭＹは、並行（コンカレント）システムの異種(heterogeneous)モデリング、シミュレーションおよび設計を研究するプロジェクトである。現在の実装は、ＪＡＶＡ（登録商標）プログラミング言語で実現されている。http://ptolemy.eecs.berkeley.eduにあるＰＴＯＬＥＭＹウェブサイトは、ＰＴＯＬＥＭＹに関するさらに詳細な情報を含む。

【０００４】ＰＯＬＩＳは、組込みシステムのハードウェア／ソフトウェアコデザインのためのソフトウェアプログラムである。ＰＯＬＩＳは、ＰＴＯＬＥＭＹのために開発されたフレームワークを利用する。http://www-cad.berkeley.edu/~polis/にあるＰＯＬＩＳウェブサイトは、ＰＴＯＬＥＭＹに関するさらに詳細な情報を含む。

【０００５】次に、本発明を理解するための適当な基礎を提供するいくつかの主題について記述する。

【０００６】一般に、軽量オペレーティングシステムは、ハードウェアとアプリケーションレベルのコードとの間の多くの障壁を除去することによって、アプリケーションが高速に動作することを可能にする。１つの直接的な結果として、システムに残る保護機構は、あるとしてもわずかとなる。このため、このような環境は、組込みあるいは専用アプリケーションに適したものにはなるが、不完全に設計された、あるいは、悪意のある結果を生じるように意図されたアプリケーションに適したものとはならない。従来、このことは、ソフトまたはハードなリアルタイム組込みコードへの、マイクロカーネルシステムの利用を制限してきた。

【０００７】現在、良好なハードウェアーソフトウェア・コシミュレーションツールに対する需要が、次の３つ

の主要な要因によって引き起こされている。

１．コンピュータシステム（ハードウェアおよびソフトウェアの両方）のサイズおよび複雑さの増大。

２．費用効果の高いＳＯＣ(system-on-a-chip)実装に対する要求。

３．これまでの投資に対する収益を最大化するようなＩＰ(intellectual property)の再利用。

【０００８】現在、一般に、上記の要因に対する有効な解決策は、ハードウェアおよびソフトウェアの両方のコンポーネントを必要とすることが認識されている。このことは、設計空間を「コデザイン」要求へと切り開いた。多くのタスクが、汎用の中央処理装置によっても、専用（あるいはプログラム可能）ハードウェアによっても実行可能である場合、最も有効な分割はどのようにしてなされるか。設計ステップは、次のようなフィードバックループで作用する。

１．タスクへの初期分割を行い、タスクアルゴリズムをコーディングする。

２．タスクをハードウェアまたはソフトウェアコンポーネントに割り当て、これらの割当てに従って適当な実行可能コードを生成する。

３．高レベルシミュレーションを行い、基本機能および動作制約を確認する。

４．設計基準を満たすように、エラーを訂正し可能な再分割を行う。

５．ハードウェアコンポーネントの完全な合成と、ソフトウェアの最適化を行う。

６．すべてのコンポーネントの低レベル（タイミングの正確な）シミュレーションを行う。

７．許容可能なあるいは最適な解に向けて反復する。

【０００９】この点まで採用されたこの手順に対する代表的なアプローチが、ＰＴＯＬＥＭＹ／ＰＯＬＩＳ環境で例証されている。ＰＴＯＬＥＭＹ／ＰＯＬＩＳは、タスクがソフトウェアあるいはハードウェアのいずれの実装で合成されることも可能にすることによって、真のコデザイン環境であることを主張している。ＰＴＯＬＥＭＹ／ＰＯＬＩＳ環境は、与えられた分割の検査を可能にする統合シミュレーションツールのためのフレームワークを提供するとともに、合成された結果を生成する最終的なコード生成フェーズを提供する。

【００１０】しかし、ＰＴＯＬＥＭＹ／ＰＯＬＩＳ環境のアーキテクチャは、その作者およびその一次ユーザ（すなわち、ハードウェアエンジニア）の先入観を反映している。ＰＴＯＬＥＭＹ／ＰＯＬＩＳ環境には、システム設計がコシミュレートされるためのソフトウェアコンポーネントに対する最小限のサポートしかない。このことは、ソフトウェア実装のためのものを含めて、ＥＳＴＥＲＥＬにおけるすべてのタスク仕様を書く必要があることから明らかである。ＥＳＴＥＲＥＬは、状態マシン型のＶＨＤＬに対するフロントエンドとしては適当な



プログラミング言語であるが、ソフトウェアのためにＣへと合成される言語としては大変である。

【００１１】プログラマの観点から見ると、「良い」ＥＳＴＥＲＥＬであっても、「悪い」Ｃを生成する。実際、ＥＳＴＥＲＥＬは、そのプログラミング様式（単純なif-thenテストと、goto分岐）において、ＦＯＲＴＲＡＮに非常に近いコードを生成する。このことは、ある程度のマシンコードを生成するだけでもＣコンパイラに重い負荷をかけ、よりグローバルな最適化を行う能力を制限する。また、この条件のため、ソフトウェアエンジニアは、既存のＩＰを捨てることを余儀なくされ、有用で試験済みの（しかも信頼された）アルゴリズムを再コーディングしなければならない。オブジェクト指向技術のサポートがないことはいうに及ばず、最新の言語の高水準の特徴がないため、ソフトウェア分割は、この階層における下層階級となる。

【００１２】最新のプログラミング言語（例えば、Ｃ＋＋）および高水準ハードウェア設計言語の両方の表現力を有する言語がないため、統一されたコデザイン環境に取り組もうとするのにさえかなりの抵抗がある。

【００１３】ＰＴＯＬＥＭＹフレームワークの別の実現も可能であり、以上の問題点の解決に役立つ可能性がある。このようなシステムは、ソースコード形式で入手不可能なこともある市販のＩＰを含む既存のコードを利用することができなければ、可能な設計空間を制限し、最適な結果を排除することになる可能性がある。（コストや市場出荷までの期間(time-to-market)の理由から）主な問題点が、ハードウェアとソフトウェアにすでに分割された既存の「レガシー」ＩＰの再利用である場合、設計空間は変わる。例えばＣコードやＶＨＤＬのような複数の方式で存在するコンポーネントが容易に思いつく。この場合、問題は次のようになる。

１．一緒に用いられる場合に設計制約を満たす既存のＩＰのセットが存在するか。

２．存在する場合、制御信号およびデータが正しく流れるように、これらのプロセスをどのように「のり付け」しなければならないか。

３．存在しない場合、まずい点はどこか。また、その点を解決するのに必要なアプローチは何か。

【００１４】これらの問題は、既存の設計どうしの間のインタフェースと、システム全体の総合的な正しい動作とを重要視し、したがって、コシミュレーションおよびコベリフィケーション(co-verification)の点を重要視したものである。

【００１５】動的再分割のことを仮に除いて考えると、プログラマは、自分の必要に最も適した言語（１つの言語または複数の言語）で自分の考えを表現する自由がある。追加の「のり(glue)」は、ハードウェアエンティティ（例えば、アドレスデコーディングロジック）やソフトウェアモジュール（例えば、ＣＰＵ外ハードウェアチ

ップのためのドライバ）として提供可能である。この段階で必要とされるのは、設計を評価するための迅速なフィードバックシステムである。

【００１６】ここでも、市販のコシミューレションツールは、約束するもののすべてを提供してはいない。ときとしては、この「ツール」は、市販のシミュレータを接続することができる単なる統合フレームワークである。これは、多くの重要な作業を節約し、これが「プラグアンドプレイ」フレームワークを提供する場合には、それぞれの要求を解決するのに最適なツールが接続されることが可能となる。しかし、個々のコンポーネントどうしの間に直接の通信がなく、したがって直接のフィードバックがない場合には、設計空間が増大すると、このアプローチは最終的に機能しなくなる可能性がある。このような実装は、現在、あるツールから別のツールにトレースを供給して収束を待つ反復的「逐次近似」法によって特徴づけられる。

【００１７】１つの代替アプローチは、要求されるフィードバックパスをすでに備えたアーキテクチャフレームワークおよびコシミュレーション環境を開発し、これに、一連の問題群全体にわたり役立つ可能性のあるターゲット変更可能(re-targetable)ツールを配置するものである。

【００１８】ハードウェアエンジニアリングの観点ではなくソフトウェアの観点からコシミュレーション問題を検討し、特に、ＳＯＣ集積の同様のアプリケーションを考慮すると、次のようないくつかの類似点が浮かび上がる。

【００１９】１ａ．軽量ソフトウェアシステムは、小規模の（ときにはリアルタイムの）スケジューラを通る相互通信プロセスのセットとして書かれる傾向がある。

１ｂ．ＶＨＤＬコードは、ワイヤのセットを通る相互通信エンティティのセットとして書かれる傾向がある。

【００２０】２ａ．ソフトウェアプロセスはイベント駆動型である傾向がある。すなわち、ソフトウェアプロセスは、何らかの他のプロセスまたは外部信号をウェイト（待機）し、何らかの処理を実行し、そして、次のイベントをウェイトする。

２ｂ．ハードウェアエンティティはイベント駆動型である傾向がある。すなわち、ハードウェアエンティティは、入力に現れる何らかの条件をウェイトし、何らかの処理を実行し、そして、次のイベントをウェイトする。

【００２１】３ａ．ソフトウェアプロセス間のイベントは通常、送信側が情報の「パケット」（データ領域）を用意し、データ同期ポイントとして作用するスケジューラを通して、「信号」を別のプロセスに送ることによって進行する。

３ｂ．ハードウェアプロセス間のイベントは通常、送信側が「データ」ワイヤのセット（例えば、バス）上に値を用意し、データ同期ポイントとして作用するグルーロジック（例えば、ラッチ）を通して、信号（例えば、チップ選択(chip-select)信号）を別のチップに送ることによって進行する。

【００２２】４ａ．シングルＣＰＵシステムでは、真のソフトウェア並列化はなく、見かけの並列化が、オペレーティングシステムがタスク間を切り換えることによって行われる。

４ｂ．きわめて自明なハードウェアを除くすべてのハードウェアでは、真の並列化があり、これは、独立のハードウェアユニットが分散クロックを受信し、それぞれそのそのクロックサイクル中に自己のタスクを実行することによって達成される。

【００２３】５ａ．適切に設計されたソフトウェアシステムでは、個々のプロセスは、あらかじめ定義されたプロセス間通信経路（これは、動的に割り当てられることも可能である）を除いては、独立に動作する。

５ｂ．適切に設計されたハードウェアシステムでは、ハードウェアプロセスの個々のインスタンスは、信号およびワイヤを用いて適切に定義された通信経路（これは、「グルーロジック」内の設定によって決定されることも可能である）を除いては、単独で動作する。

【００２４】ＶＨＤＬコードのシミュレーションのためのツール（例えば、Model Technologies社のＶＳＩＭ）がすでに存在する。このようなツールは、システム全体の内部の個々のエンティティの挙動をエミュレートすることによって動作し、上記の項目５ｂに基づいて、ハードウェア並列化の代わりにソフトウェア型の並列化（すなわち、順次実行）を行う。これは、同期ポイントにおける個々のプロセスの状態（例えば、デルタあるいはクロック遷移）をそれぞれチェックし、各シミュレーションサイクルの終端で状態変化を行うことによってなされなければならない。通常、シミュレーション環境全体が、ワークステーション上の単一プロセスとして動作し、内部スケジューラが、その内部のＶＨＤＬプロセスの状態を追跡する。

【００２５】図１に、従来のＶＨＤＬシミュレーションを例示する。通常、シミュレーション環境は、マルチプロセシングオペレーティングシステム（例えばＵＮＩＸ（登録商標）１）内で実行される。シミュレーションのコンポーネントは、命令セットシミュレータ４、ＶＨＤＬシミュレーション５、および、信号を受渡しするためのインタフェース６である。ＶＳＩＭ環境２は、内部スケジューラ３を用いて、シミュレーションコンポーネントを制御する。図１に示されているように、割込み７は、ＶＨＤＬシミュレーション５から、インタフェースを通じて、命令セットシミュレータに渡されなければならない。外部コンポーネント８との相互作用は、内部スケジューラ３を通してＶＳＩＭ環境２に渡されなければならず、さらに、ＶＳＩＭシミュレータ２は、ＵＮＩＸシステム１を通じてデータを渡さなければならない。オ

ペレーティングシステムにとってＶＳＩＭ環境２の全体がＵＮＩＸオペレーティングシステムシェル１で実行される単一のプロセスとして見えるため、内部スケジューラ３は、従来のシミュレーションシステムの本質的コンポーネントである。同時に、このプロセスが、システムの他の外部コンポーネント８と相互作用することを可能にするため、ＵＮＩＸオペレーティングシステム１のマルチプロセシングサポートが要求される。

【００２６】現実のシステムは、単純な「部分の総和」アプローチが示すようなものよりもしばしばはるかに複雑であり、システム統合の問題は、個々の設計フェーズ中に見落とされる設計欠陥を明らかにすることがある。このことは特に、非決定性システム（例えば、ベストエフォート型ネットワークリンクを通じて受信されるデータの処理）の設計の際には明らかである。また、設計空間は、起こりうるさまざまなエラー（パケット損失、データ破損、リアルタイムデッドライン超過など）と、それが実際に起きたときにエラーを処理するストラテジとを考慮に入れなければならない。

【００２７】実世界のアプリケーションにおけるもう１つの問題点は、データの周期性が、マイクロ秒やナノ秒ではなく、数分の１秒以上となることが多いことである。例として、ＭＰＥＧデータの表示システムを考える。（毎秒３０フレームの表示の場合）３３ｍｓという基本フレーム周期があるが、これは、データの真の周期ではない。ＭＰＥＧは、データの（単一の）フルフレームと、それを修正する一連の補間フレームとを有するＧＯＰ(group of pictures)セグメントへと、フレームをクラスタ化する。通常、ＧＯＰは、１５フレーム、すなわち、０．５秒のオーダーのデータである。フルフレームデータの損失は実質的にそのＧＯＰ内の後続フレームを無意味にするため、このレイヤはエラーの伝搬を制限する。

【００２８】したがって、このようなシステムの設計においては、１個のＧＯＰ全体のデータより短いシミュレーションは、不正なエラー処理状況を見落とす可能性が高い。実際、秒のオーダーのデータと、１分に達するシミュレートされた実行を要求しない試験ストラテジを考えることは困難である。このようなシステムでは、任意の設計に対して、少なくとも２つの「予備」ステップがあることになる。第１のステップは、その設計がフルスピードＭＰＥＧデータのリアルタイム制約を処理することの確認である。第２のステップは、ずっと長い期間にわたるエラー条件をどの程度うまく処理するかについて調べることである。

【００２９】

【発明が解決しようとする課題】より強力な（例えば、コスト、エネルギーに関して）ＣＰＵと、追加のカスタムハードウェア（例えば、ＦＰＧＡやＡＳＩＣ）の間には一般的なトレードオフがある。上記のＭＰＥＧの例を用いると、ＭＩＰＳプロセッサでハードウェア支援がない場合、１０フレーム／秒を超える表示は不可能である。しかし、インテルＭＭＸレベルのプロセッサの場合、追加ハードウェアがなくても、３０フレームを優に超える表示が可能である。このような観察は、ハードウェアおよびソフトウェアの両方に対して単一の設計言語を使用することに対する強力な反論となる。というのも、このパフォーマンスは、主要なルーチンをインテルのアセンブリ言語で直接にコーディングすることによってのみ達成することができるからである。当面する問題を正しく解決することが可能な機能を無視することによって、候補となる可能性のあるＣＰＵを「不自由にする」のはほとんど無意味である。

【００３０】

【課題を解決するための手段】本発明は、上記の状況に鑑み、従来技術の上記の問題点と制限を克服するものである。

【００３１】本発明のさらに他の利点については、その一部は以下の説明に記載されており、一部はその説明から明らかであるか、または、本発明を実施することにより知ることができる。本発明の利点は、特許請求の範囲に特に記載された手段と組合せによって実現され達成される。

【００３２】本発明の第１の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーション（協調検証）を行う方法が実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。この方法は、軽量コンピュータシステムオペレーティング環境をターゲットとする構文(construct)からなるハードウェアモデルに、電子回路の動作シミュレーションを翻訳することを含む。さらに、この方法は、サイクル精度の命令セットシミュレーションを構成することを含む。これにより、命令セットシミュレーションは、電子回路をターゲットとする制御プログラムの一部を実行する。さらに、この方法は、電子回路のハードウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成することを含む。さらに、この方法は、試験対象システムに刺激を入力することにより、試験対象システムに結果を出力させることを含む。

【００３３】本発明の第２の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。このコンピュータシステムは、すべてのグローバル信号の状態を管理するソフトウェア手段を有する。さら

に、このコンピュータシステムは、クロック信号を生成するソフトウェア手段を有する。さらに、このコンピュータシステムは、イベントをウェイトするプロセスキューを管理するソフトウェア手段を有する。さらに、このコンピュータシステムは、所定のタイミング間隔で少なくとも１つのプロセスを実行するソフトウェア手段を有する。さらに、このコンピュータシステムは、中央プロセス制御を行うソフトウェア手段を有する。

【００３４】本発明の第３の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。このコンピュータシステムは、プロセッサと、コンピュータシステムが中央制御プロセスを実行することを可能にするように適応したソフトウェア命令を含むメモリとを有する。中央制御プロセスは、複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスとを有する。さらに、中央制御サブプロセスは、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、所定のタイミング間隔を生成するタイマサブプロセスとを有する。さらに、中央制御サブプロセスは、少なくとも信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御するコントローラサブプロセスを有する。

【００３５】本発明の第４の特徴によれば、マイクロプロセッサによって制御される電子回路と、そのマイクロプロセッサをターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムが実現される。ここで、電子回路、ターゲットマイクロプロセッサおよび制御プログラムは、所定のコンピュータ言語および所定のインタフェースライブラリを用いてシミュレートされる。このコンピュータシステムは、電子回路のソフトウェアモデルを有する。電子回路の機能は、所定のインタフェースライブラリの一部を含む所定のコンピュータ言語構文に翻訳されている。さらに、このコンピュータシステムは、ターゲットマイクロプロセッサの命令セットシミュレーションを有し、これにより、命令セットシミュレーションは、制御プログラムの一部を実行するときにターゲットマイクロプロセッサによって実行される命令をシミュレートする。さらに、このコンピュータシステムは、電子回路のソフトウェアモデルと、ターゲットマイクロプロセッサの命令セットシミュレーションを互いに結合する中央制御プロセスを有する。中央制御プロセスは、複数のグローバル信号の状態を管理する信号メンテナサブプロセスと、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロック信号ジェネレータサブプロセスとを有する。さらに、中央制御サブプロセスは、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブプロセスと、所定のタイミング間隔を生成するタイマサブプロセスとを有する。さらに、中央制御サブプロセスは、少なくとも信号メンテナサブプロセス、クロック信号ジェネレータサブプロセス、キューメンテナサブプロセスおよびタイマサブプロセスの実行を制御する制御サブプロセスを有する。

【００３６】本発明の第５の特徴によれば、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションを行うコンピュータシステムのための実行可能プログラムが実現される。ここで、電子回路および制御プログラムは、所定のコンピュータ言語を用いてシミュレートされる。この実行可能プログラムは、複数のグローバル信号の状態を管理する第１実行可能コード部分を有する。さらに、この実行可能プログラムは、複数のクロック信号を生成する第２実行可能コード部分を有する。さらに、この実行可能プログラムは、イベントをウェイトするプロセスキューを管理する第３実行可能コード部分を有する。さらに、この実行可能プログラムは、所定のタイミング間隔を生成する第４実行可能コード部分を有する。さらに、この実行可能プログラムは、少なくとも第１、第２、第３および第４実行可能コード部分の実行を制御する第５実行可能コード部分を有する。

【００３７】本発明の第６の特徴によれば、電子回路とその電子回路を制御する制御プログラムとを含むシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法が実現される。この方法は、メモリとキャッシュの間の相互作用をデータバス上の信号のシーケンスとしてモデル化することを含む。ここで、ターゲットプロセッサは、メモリオペレーションの完了を待機することによってキャッシュがロードされるまでストールされる。さらに、この方法は、ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いるとともに、内部バス幅およびタイミングを用いて、命令パイプラインを充填して、適当なクロックサイクル数だけ遅延を行うことを含む。さらに、この方法は、ターゲットプロセッサの命令デコードサイクルを実行して、実行のために利用可能な命令を解釈することを含む。さらに、この方法は、スケジューリングされた命令がパイプライン切断をインプリメントしているかどうかを判定し、スケジューリングされた命令がパイプライン切断をインプリメントしている場合、将来の命令のためにスケジューラをス

トールさせることを含む。さらに、この方法は、スケジューリングされた命令を、適当な命令パイプラインまたはハードウェアコンポーネントに転送し、各命令ごとにサイクル実行時間を計算することを含む。さらに、この方法は、非決定性タイミングがあるかどうかを判定する。さらに、この方法は、計算された実行サイクルの終端で利用可能な結果を出力することを含む。さらに、この方法は、ターゲットプロセッサのエミュレートされた制御レジスタとともに、信号インタフェースを用いて、割込みハンドラが次のサイクルにスケジューリングされるべきかどうかを決定することを含む。

【００３８】本発明の上記の特徴およびその他の利点は、以下の詳細な説明から、また、添付図面を参照して、明らかとなる。

【００３９】

【発明の実施の形態】本発明の特徴について説明する前に、本発明の理解を助けるため、および、さまざまな用語の意味を提示するために、従来技術に関していくつかの詳細な事項について説明する。

【００４０】本明細書において、「コンピュータシステム」という用語は、可能な限り最も広い意味を含み、独立（スタンドアローン）のプロセッサ、ネットワーク化されたプロセッサ、メインフレームプロセッサ、およびクライアント／サーバ関係にあるプロセッサを含むが、これらに限定されない。「コンピュータシステム」という用語は、少なくともメモリおよびプロセッサを含むものと理解されるべきである。一般に、メモリは、いろいろなときに、実行可能プログラムコードの少なくとも一部を格納し、プロセッサは、その実行可能プログラムコードを構成する命令を実行する。

【００４１】本明細書において、「組込みコンピュータシステム」という用語は、組込み中央プロセッサとオブジェクトコード命令を有するメモリとからなるが、これに限定されない。組込みコンピュータシステムの例には、携帯情報端末（ＰＤＡ）、セルラ電話機およびディジタルカメラがあるが、これらに限定されない。一般に、どんなに原始的であっても、中央プロセッサを用いてその機能を制御する機器は、組込みコンピュータシステムを有するということができる。組込み中央プロセッサは、メモリに格納されたオブジェクトコード命令を実行する。組込みコンピュータシステムは、キャッシュメモリ、入出力デバイスおよびその他の周辺装置を有することが可能である。

【００４２】理解されるように、「所定オペレーション」という用語および「コンピュータシステムソフトウェア」という用語は、本明細書の目的では、実質的に同じものを意味する。本発明の実施にとって、メモリおよびプロセッサが物理的に同じ場所に位置する必要はない。すなわち、プロセッサおよびメモリが物理的に異なる装置にあることや、地理的に異なる場所にあることも予期される。

【００４３】本明細書において、当業者には理解されるように、「媒体」あるいは「コンピュータ可読媒体」は、ディスケット、テープ、コンパクトディスク、集積回路、カートリッジ、通信回線を通じてのリモート伝送、あるいはその他の同様の、コンピュータが使用可能な媒体を含むことが可能である。例えば、コンピュータシステムソフトウェアを分散させるため、供給業者（サプライや）は、ディスケットを提供することも可能であり、また、衛星伝送、直接電話リンク、あるいはインターネットを通じて何らかの形で所定オペレーションを実行する命令を伝送することも可能である。

【００４４】コンピュータシステムソフトウェアは、ディスケットに「書き込まれる」ことも、集積回路に「格納される」ことも、通信回線を通じて「伝送される」ことも可能であるが、理解されるように、本明細書の目的では、コンピュータ使用可能媒体は、所定オペレーションを実行する命令を「有する」ということにする。すなわち、「有する」という用語は、所定オペレーションを実行するための命令がコンピュータ使用可能媒体に関わる上記およびすべての等価な方法を含むことを意図するものである。

【００４５】したがって、簡単のため、「プログラム製品」という用語は、以下、任意の形式で所定オペレーションを実行するための命令を有する（上記で定義）コンピュータ使用可能媒体を指すために用いられる。

【００４６】次に、添付図面を参照して、本発明の特徴の詳細な説明を行う。

【００４７】ＶＨＤＬプロセスと、軽量オペレーティングシステムで実行されるプロセスとの間の顕著な類似性のため、ＶＨＤＬプロセスを、個別のソフトウェアプロセスとして、軽量オペレーティングシステムを用いて書き、イベントをスケジューリングすることが可能となる。好ましくは、軽量オペレーティングシステムは、メッセージによるプロセス間通信を提供する小さいマイクロカーネルをもとに構築される。メッセージは、関連するパラメータを有するオペレーションコードであり、上記の項目３ａのイベント駆動モデルを表現する。好ましくは、メッセージは、プロセス間で受渡しされるシステム構造体にマッピングされるグローバルデータ定義である。イベントのセマンティクスはアプリケーションによって定義される。この場合、プロセス間メッセージは、ＶＨＤＬ信号値の変化についてプロセスに通知するために用いられることも可能である。好ましくは、軽量オペレーティングシステムは、信号線、アドレスバス、データバスおよびクロック分配方式が少数の基本メッセージから構築されることを可能にする一群のメッセージプリミティブを提供する。好ましくは、マイクロカーネルコアによって供給されるメッセージに加えて、プロセスは、アプリケーション固有のインタフェースをインプリ

17

メントするために自己のメッセージコードおよびパラメータを定義することも可能である。その場合、このインタフェースは、共有ライブラリにインプリメントされたルーチンコールのセットにまとめられる。

【００４８】次に、図２を参照して、電子回路と、その回路をターゲットとするソフトウェアとのコバリデーションを行う本発明の特徴について簡単に説明する。電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。Ｓ０９００で、１つまたは複数の回路記述言語によってモデル化された、コンピュータハードウェアの動作の要素を、選択された軽量オペレーティングシステムの構文およびメソッドに翻訳する。Ｓ１０００で、電子回路設計の動作記述を、軽量コンピュータシステムオペレーティング環境をターゲットとする前記構文からなるソフトウェアモデルに翻訳する。Ｓ１１００で、電子回路設計をターゲットとする制御プログラムの部分を命令セットシミュレーションが実行するように、命令セットシミュレータを所定のコンピュータ言語で構成する。Ｓ１２００で、電子回路設計のソフトウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象システムを作成する。最後に、Ｓ１３００で、試験対象システムに刺激を入力することにより、試験対象システムに結果を出力させる。

【００４９】次に、電子回路および制御プログラムのコバリデーションを行う本発明の特徴についてさらに詳細に説明する。

【００５０】ステップＳ０９００では、ハードウェア記述言語を調べる。好ましくはＶＨＤＬ言語がモデル化されるが、本発明の方法および応用は、特定の言語や特定の軽量オペレーティングシステムに限定されない。動作レベルＶＨＤＬの制御フローおよび様式は、以下の具体的な点でＣに容易にマッピングすることが可能である。

【００５１】ＶＨＤＬの変数はＣの変数に似ている。その値は、制御がプロセスを離れて戻ってきたときにも保持されている。しかし、その名前のスコープは、それがインスタンス化されたプロセスについてローカルである。したがって、本発明は、ＶＨＤＬの変数をＣのスタティック変数で置き換える。ビットベクタ型は、３２ビット幅（３２ビットプロセッサで動作する処理系の場合）までは、Ｃの符号なし整数型に含めることができる。個別ビットアクセスおよびスライス演算はいずれも、シフトおよびマスク演算で実行することができる。

【００５２】信号は、プロセス間で値を送信するために用いられる。信号の値のスコープは、システム全体を通じてグローバルであるが、その名前は、特定のプロセスについてローカルである。本発明では、プロセスの複数のコピーを、それぞれ相異なる信号（例えば、UART_A_Chip_Select、UART_B_Chip_Select）にアクセスする同一

18

のローカル信号名（例えば、Chip_Select）でインスタンス化することが可能である。「０」や「１」のような単純な信号値と、スライス値とは、マシン内のビットパターンとして容易に表現される。ＶＨＤＬは、用いられる電気的モデルに依存して、「Ｚ」や「Ｗ」のようなこの範囲外の論理値をサポートする。

【００５３】個々のＶＨＤＬプロセスは、一定のナノ秒数の間、または、その信号に関する条件のセットが満たされるまで、その実行をサスペンドすることが可能である。ＶＨＤＬプロセスが「動作可能」であるための条件を決定することは、言語構文のマッピングにおいてインプリメントされなければならない機能の１つである。

【００５４】また、ＶＨＤＬ言語は、ハードウェアの実行のフローを制御する構文を提供する。これは、軽量オペレーティングシステムによって用いられるプログラミング言語（この場合はＣ）の等価な構文にマッピングされなければならない。

【００５５】好ましくは、シミュレーションシステム内の中央プロセスは、各信号の値を追跡すること、および、その値をモニタするプロセス間に信号変化を分配することの処理を行う。以下の説明でこのプロセスに言及するときには、「ファブリック」プロセスと呼ぶことにする。本発明は、軽量オペレーティングシステムのメッセージ受渡しオペレーションを用いて、プロセス間で、信号値の変化やその他の構文の受渡しを行う。例えば、"wait for N ns"（Ｎナノ秒ウェイト）というＶＨＤＬ構文は、次のような２つのユーザ定義パラメータを有するメッセージにより表現される。

Msg WAITFOR {int CLOCKT; int TICKS;}

TICKSパラメータは、ＶＨＤＬにおけるＮパラメータからナノ秒数を提供し、CLOCKTパラメータは、クロック相対同期ポイントを提供する。

【００５６】メッセージインタフェースは、アプリケーションコードに直接には現れない。すべての相互作用は、マイクロカーネルコア機能を呼び出す共有ライブラリルーチンを通じてなされる。

【００５７】信号は文字列を用いて命名されるが、opaque型ポインタを用いて参照される。以下のインタフェースルーチンは、信号名と参照の間のマッピングをインプリメントする。さらに、このインタフェースは、信号値の変化について中央制御プロセッサに通知し、信号の値の変化をウェイトする、共有ライブラリルーチンを含む。

【００５８】・void *fabinit(void)：　このルーチンは、中央制御プロセスインタフェースライブラリを初期化する。このルーチンは、信号メンテナを使用することになる各プロセスによって１回呼び出されなければならない。

【００５９】・void *fabric_find_signal(char *SIGNAL)：　このルーチンは、（グローバルに命名された）SI

19

GNALパラメータを参照するために使用可能なopaque型ポインタを返す。このルーチンは、すべての登録された名前に対する文字列比較を実行し、信号参照が存在すればそれを返す。信号がリスト内にまだない場合、初期値０で作成される。このようにして、プロセスは、特定の順序でインスタンス化あるいは初期化されることは不要である。

【００６０】・uint fabric_read_signal(void *SYM)：　このルーチンは、opaque型SYMパラメータによって参照される信号の現在値を、もしそれが定義された（すなわち、非浮遊トライステート）値を有していれば、返す。

【００６１】・void fabric_set_clock(char *NAME, int INITIAL, int PERIOD)：　このルーチンは、このプロセスに対する同期オペレーションで用いられるクロック信号を定義する。このルーチンは、ファブリックプロセス内への共有ライブラリ呼出しであり、ファブリックプロセスがすべての遷移を内部的に処理する。INITIALパラメータは、信号の開始値（通常０）を指定し、PERIODパラメータは、遷移間の間隔（すなわち、クロック周期の半分）を質する。これは信号を定義するため、クロックはグローバルであり、複数のプロセスが同じくロックを共有することができる。

【００６２】・void fabric_tristate(void *SYM, int VALUE)：　このルーチンは、ＩＥＥＥ論理システムの拡張値の設定およびそれとの比較を可能にする多値論理信号をサポートする。

【００６３】・void fabric_write_signal(void *SYM, uint VALUE)：　このルーチンは、opaque型ポインタSYMによって参照される信号の値をVALUEにセットする。

【００６４】ファブリックプロセスは、信号が、単純な２値ドメイン内で作用しているか、それとも、多値論理を使用しているかを知らなければならない。このドメインは、システムにわたり信号変化を送信するために用いられるインタフェースルーチンによって決定され、処理系は、多値論理ドメインが表現されることを可能にする。それぞれの多値論理ドメインに要求される拡張値の表現を含む個々のＣヘッダファイル（例えば、ieee.h）は、個々の論理ドメインをカプセル化することが可能である（例えば、std_logicパッケージ）。

【００６５】以下の形のWaitルーチンはすべて、CLOCKTパラメータに次の３つの値のうちの１つをとる。０は、非同期オペレーションを意味する（指定された条件が満たされるとすぐにプロセスをスケジューリングする）。－１は、条件がプロセスのクロック信号の立下り端で満たされる場合にプロセスをスケジューリングし、＋１は、条件がプロセスのクロック信号の立上り端で満たされる場合にプロセスをスケジューリングする。

【００６６】・void WaitClk(int CLOCKT)：　このルーチンは、（CLOCKTパラメータに従って）そのクロックの

20

次の立上りまたは立下り端まで、現在のプロセスをサスペンドする。

・void WaitForc(uint TICKS, int CLOCKT)：　このルーチンは、TICKS個のクロックティックの間、現在のプロセスをサスペンドする。

・void WaitUntilc(void *SYM, int OP, uint VALUE, int CLOCKT)

・void WaitUntil2c(void *SYM1, int OP1, uint VALUE1, void *SYM2, int OP2, uint VALUE2, int CLOCKT)

・void WaitUntil3c(void *SYM1, int OP1, uint VALUE1, void *SYM2, int OP2, uint VALUE2, void *SYM3, int OP3, uint VALUE3, int CLOCKT)：　これらのルーチンは、１、２または３個の条件のうちのいずれか１つ以上が（同時に）真になるのをウェイトする。それぞれの条件は、信号SYM、オペレーションOPおよびVALUEである。例えば、"EQ"オペレーションは、SIGNALとVALUEの間の等値性をテストするために用いられる。

【００６７】上記のルーチンは、プロセス間メッセージを形成し、それをファブリックプロセスに送る。このコントローラプロセスは、信号に対するすべての変化を追跡し、イベントをウェイトしているプロセスをいつ呼び起こすかを決定する。

【００６８】これらのルーチンは、共有ライブラリにインプリメントされ、Ｃヘッダファイルを通じてエクスポートされた。また、ヘッダファイルは、ファブリックプロセスによって管理されるグローバルナノ秒ティックカウンタもエクスポートするため、個々のプロセスからのトレース出力は、現在のタイムスタンプでタグ付けされることが可能である。

【００６９】図３を参照すると、本発明は、それぞれの信号名を前処理する。Ｓ１０１０で、それぞれの信号に対して、本発明は、名前にプレフィクス句"signal_"を付けることによって、内部変数を生成する。したがって、信号CHIP_SELECTへのopaque型ポインタは"static void *signal_CHIP_SELECT"となる。その後、この新しい信号名は、プロセスがインスタンス化されるときに、信号参照に結合される。インスタンス化された信号名がその信号名と同一である必要はない。例えば、signal_CHIP_SELECTは次のようにインスタンス化されることも可能である。

signal_CHIP_SELECT=fabric_find_signal("UART_A_CHIP_SELECT");

ＶＨＤＬの本体で、プリプロセッサは、信号CHIP_SELECTへの参照を、変数signal_CHIP_SELECTを参照するように修正する。インスタンス化されるとき、変数signal_CHIP_SELECTは、UART_A_CHIP_SELECT信号に結合される。

【００７０】Ｓ１０１５で、前処理すべき信号名がさらにあるかどうかを判定する。ある場合、Ｓ１０１８で、前処理すべき次の信号名を取得し、プロセス制御フローはＳ１０１０に戻って、信号名の前処理を続ける。ない

場合、プロセス制御フローはＳ１０２０に進む。

【００７１】図３を参照して、プロセス制御フローがＳ１０２０～Ｓ１０６０に到達すると、どのタイプの動作言語構文が用いられているか、および、その動作言語を、軽量コンピュータオペレーティングシステムをターゲットとする構文にどのように翻訳するかについて判定を行う。好ましくは、使用される動作言語はＶＨＤＬであり、翻訳のために用いられる構文はＣベースのものである。

【００７２】Ｓ１０２０で、翻訳されるべき動作言語構文が内部変数への代入であるかどうかを判定する。内部変数への代入である場合、プロセス制御フローはＳ１０２５に進む。内部変数への代入でない場合、プロセス制御フローはＳ１０３０に進む。

【００７３】Ｓ１０２５で、ＶＨＤＬ代入からＣ代入への翻訳が次のように行われる。

ＶＨＤＬ：　i_word := 1;

Ｃ：　static uint i_word;

i_word = 1;

この翻訳の終了後、プロセス制御フローはＳ１０２０に戻る。

【００７４】Ｓ１０３０で、翻訳されるべき動作言語構文が外部変数への代入であるかどうかを判定する。外部変数への代入である場合、プロセス制御フローはＳ１０３５に進む。外部変数への代入でない場合、プロセス制御フローはＳ１０４０に進む。

【００７５】ほとんどのインタフェースルーチンは、ＶＨＤＬ－Ｃ翻訳が読みやすくかつ理解しやすくなるように設計されたプリプロセッサマクロを通じてアクセスされる。これらのマクロは、信号名と変数名の間の変換を処理する。以下のマクロにおいて、"signal"はＶＨＤＬ信号の名前を意味する。これらの名前は、上記のopaque型参照へと前処理される。

・uint GetSignal(signal NAME)：　このマクロは、NAMEという信号の現在値を返す。

・void Signal(signal NAME, uint VALUE)：　このマクロは、NAMEという信号のVALUEをその（３２ビットの）値にセットする。

・void SignalZ(signal NAME)：　このマクロは、NAMEという信号の値をＩＥＥＥのＺ値にセットする。

【００７６】Ｓ１０３５で、ＶＨＤＬ代入から外部信号への翻訳が次のように行われる。

ＶＨＤＬ：　a_busp <= '0';

マクロ：　Signal(a_busp, 0);

Ｃ：　fabric_write_signal(signal_a_busp, 0);

生成される代入は、ランタイムインタフェースを通じてファブリックプロセスへの呼出しを生成することになる。この翻訳の終了後、プロセス制御フローはＳ１０２０に戻る。

【００７７】Ｓ１０４０で、翻訳されるべき動作言語構文がウェイトポイントであるかどうかを判定する。ウェイトポイントである場合、プロセス制御フローはＳ１０４５に進む。外部変数への代入でない場合、プロセス制御フローはＳ１０５０に進む。外部信号への代入と同様に、ウェイトポイントは、次のようなプリプロセッサマクロを用いたインタフェース呼出しにマッピングされる。



・void WaitClk |FIR| (void)：　これらのマクロはそれぞれ、プロセスクロック信号の次の立上りまたは立下り端をウェイトする。

・void WaitFor |FIR| (uint TICKS)：　これらのマクロは、TICKSナノ秒間ウェイトする。プロセスは、そのティックにおける、または、次の適当なクロック遷移における実行のためにスケジューリングされる。

・void WaitUntil |FIR| (signal NAME, int OP, uint VALUE)

・void WaitUntil2 |FIR| (signal NAME, int OP, uint VALUE, signal NAME2, int OP2, uint VALUE2)

・void WaitUntil3 |FIR| (signal NAME, int OP, uint VALUE, signal NAME2, int OP2, uint VALUE2, signal NAME3, int OP3, uint VALUE3)：　これらのマクロは、ある条件が１、２、または３個の信号に現れるまでプロセスがウェイトすること、および、要求されるクロック同期に従ってプロセスがスケジューリングされることを可能にする。

【００７８】Ｓ１０４５で、ＶＨＤＬ構文の翻訳は次のように行われる。

ＶＨＤＬ：　wait until (startn = '0');

マクロ：　WaitUntil(startn, EQ, 0);

Ｃ：　WaitUntilc(signal_startn, EQ, 0, 0);

ウェイトポイントの翻訳の終了後、プロセス制御フローはＳ１０２０に戻る。

【００７９】Ｓ１０５０で、ＶＨＤＬ制御構文が、等価なＣ言語オペレーションに翻訳される。この翻訳は、追加のインタフェース呼出しを含むことがある。例として、'if'制御構文の翻訳は次のように行われる。

ＶＨＤＬ：　if (resetn = '0') then

マクロ：　if (GetSignal(resetn) == 0)　|

Ｃ：　if (fabric_read_signal(signal_resetn) == 0)　|

制御構文の翻訳の終了後、プロセス制御フローはＳ１０２０に戻る。

【００８０】図４を参照すると、Ｓ１０６０で、翻訳する必要のある動作言語構文がさらにあるかどうかを判定する。さらにある場合、プロセス制御フローはＳ１０２０に進む。さらにない場合、変換プロセスは完了する。

【００８１】これらの少数の基本変換により、動作ＶＨＤＬコードをＣに翻訳し、そのコードをコンパイルし、軽量オペレーティングシステム用にそれをリンクすることが可能である。ＣとＶＨＤＬの間のマッピングは一対

一であり可逆であるため、もとの設計がＶＨＤＬで定式化されているか、それとも、翻訳可能Ｃで定式化されているかで、シミュレーションには差がない。翻訳が正しく実行される限り、結果は同じになる。この翻訳から生じるＣのサブセットは依然として強力なプログラミング言語である。これは、ポインタ間接参照や高度なデータ構造体のような特にＣＰＵベースのアルゴリズムを目的とするいくつかの機能を欠いているが、それ以外の点では、ターゲット変更可能設計の出発点として働くことが可能である。分割がわかったら、個々の解（ハードウェアまたはソフトウェア）の機能を十分に活用すべきである。

【００８２】本発明の特徴によれば、設計のソフトウェアコンポーネントは、ターゲットプロセッサの命令セットシミュレータ（ＩＳＳ：Instruction-Set Simulator）を用いてシミュレートすることが可能である。一般に、このようなシミュレータは、次の３つの要素を有する。

１．ターゲットプロセッサのオペレーションおよびレジスタのエミュレーション。

２．エミュレートされるプロセッサの状態を表示し、エミュレータのオペレーションを制御する、外部プログラムへのインタフェース、あるいはヒューマンインタフェース。

３．試験対象システムに存在するデバイスを表す外部ハードウェアのエミュレーションへのインタフェース。

【００８３】シミュレータの高度化と、外部デバイスとの相互作用とに依存して、結果は、試験対象システムの基本機能のみを確認することもあり、あるいは、すべてのプロセスのサイクル精度のシミュレーションを与えることもある。非サイクル精度のＩＳＳの構成を、本発明の第１の特徴として以下で説明し、本発明のさらに他の特徴による変形についてはその後説明する。

【００８４】要素１は、ソフトウェアが、指定されたアーキテクチャによるプロセッサのレジスタをエミュレートすることを要求する。このようなレジスタには、汎用レジスタ、浮動小数点レジスタ、制御レジスタ、および、ユーザソフトウェアから直接にはアクセス可能でない「隠れた」レジスタが含まれるが、これらに限定されない。さらに、レジスタに対するオペレーションもまた、そのようなオペレーションによって引き起こされるステータスおよびエラー情報の生成を含めて、正しくエミュレートされなければならない。

【００８５】要素２は、外部エージェントへのインタフェースを要求する。好ましくは、ＩＳＳは、非常に単純なコマンドラインインタフェースを有し、使用されるコマンドは、図５に示したテーブル１にリストされている。

【００８６】要素３は、プロセッサに接続されたメモリおよびその他の外部デバイスのエミュレーションを要求する。ＩＳＳは、２つの方法のうちの一方を用いてローカルＲＡＭをエミュレートする。ＲＡＭの量が小さい場合、ホストマシンのローカルメモリが用いられる。そうでない場合、ディスクスペースを用いて、エミュレートされるＲＡＭを保持する。これにより、実際のマシンより多くの物理メモリを有するマシンのエミュレーションが可能となる。また、これは、障害時のメモリの再検討可能なチェックポイントを提供する。予想されるように、外部ディスクを用いると、シミュレーションはかなり遅くなる。

【００８７】ＩＳＳが外部ハードウェアに応答するためには、そのハードウェアのモデルにアクセスすることができなければならない。本発明の第１の特徴によれば、外部ハードウェアの動作を表すモジュールのセットが、ＩＳＳ実行可能ファイルにリンクされる。外部デバイスは、外部グルーロジックを通じてＣＰＵアドレス空間にマッピングされると仮定され、ある範囲の（非キャッシュ）メモリアドレスがデバイスへのアクセスを引き起こすようになる。このアクセス範囲は、ハードウェアコマンドによって定義される。

【００８８】例えば、一般的なＵＡＲＴインタフェースは、次の５つの機能を有することが可能である。

【００８９】・void uart_reset(void)：　このルーチンは、ハードウェアリセット信号の送信をシミュレートするために、ＩＳＳ初期化中に呼び出される。

【００９０】・uint uart_fetch(int ADDRESS, int LENGTH)：　このルーチンは、プロセッサからデバイスへのリード（読出し）要求をシミュレートする。デバイスは、その入力アドレスライン上にADDRESSパラメータが提示されたかのように、LENGTHパラメータによって指定される適当な数のビットを返さなければならない。

【００９１】・void uart_store(int ADDRESS, int LENGTH, uint VALUE)：　このルーチンは、プロセッサからデバイスへのライト（書込み）要求をエミュレートする。このルーチンは、ADDRESSパラメータがその入力アドレスラインに提示され、かつ、VALUEパラメータがその入力データラインに提示されたかのように、作用しなければならない。LENGTHパラメータは、有効な値のバイト数を指定する。

【００９２】・void uart_tick(void)：　このルーチンは、プロセッサクロックティックごとに呼び出され、データストリーミングやタイマオペレーションをシミュレートするために使用可能である。

【００９３】・ハードウェアコンポーネントがプロセッサへの割込み信号を発生することを可能にする割込み発生ルーチン。

【００９４】このエミュレーションは、タイミング情報を含まない。ＵＡＲＴは、イベントが提示されるのと同じクロックティック内に応答している。正確なタイミングが問題とはならない試験ソフトウェアの場合、この単

純なモデルで十分であった。

【００９５】プロセッサは数百メガヘルツのクロックで動作することが可能であるが、このような速度は、命令およびデータがいずれも高速のオンチップキャッシュから来るときにのみ実現される。遅いキャッシュや外部メモリにアクセスすると、ＣＰＵの見かけのパフォーマンスは低下することになる。さらに、多くのコデザイン問題の場合、外部ハードウェアは、オフチップＲＡＭにのみ書込みが可能である。このこともまた、ソフトウェア設計に対する制約となる。このようなデータにアクセスするにはデータキャッシュをバイパスしなければならない（結果としてそのデータはキャッシュにロードされるかもしれないが）からである。もちろん、メモリマップドＩ／Ｏロケーションへのアクセスもまた、非キャッシュアクセスを通じて行われなければならない。

【００９６】しかし、このインタフェースは、コンポーネントに対する異なるアクセス時刻をエミュレートしない。すべてのメモリトランザクションは、単一サイクルで完了すると仮定される。

【００９７】基本インタフェースは、必要に応じて個々の命令エミュレーションから呼び出される次の３つのルーチンを通じて提供される。

【００９８】・void fetch(iblock *IN, uint ADDRESS1, int NBYTE, uint *ADDRESS2)：このルーチンは、（エミュレーション）アドレスADDRESS1からNBYTEバイトをロードし、それを（マシン）アドレスADDRESS2に格納（ストア）する。命令ブロックINは、リソーススコアボーディングのために用いられる。このルーチンは、キャッシュ可能領域ではデータキャッシュ探索を行い、それ以外ではエミュレートされたＲＡＭやデバイスにアクセスする。

【００９９】・void ifetch(uint ADDRESS1, int MAX, int IX)： このルーチンは、命令フェッチアルゴリズムをインプリメントする。これは、（エミュレーション）アドレスADDRESS1から始まるMAX個までのワードを、インデックスIXから始まる（４ワード）命令パイプにロードする。このルーチンは、まず、命令キャッシュを探索した後、データが見つからない場合に８ワードキャッシュラインを充填するためにメモリに進む。

【０１００】・void store(block *IN, uint ADDRESS, int NBYTE, uint *VAL)： このルーチンは、（マシン）アドレスVALから（エミュレーション）アドレスADDRESSにNBYTEバイトを書き込む。また、このルーチンは、キャッシュ可能メモリのためにデータキャッシュを更新する。命令ブロックポインタINは、スコアボーディングのために用いられる。また、これらのルーチンは、キャッシュヒット数、キャッシュミス数、および非キャッシュメモリへのアクセス数についての統計を収集する。

【０１０１】タイミング測定が不要な場合、ＣＰＵの最も単純（かつ最も高速）なシミュレーションは、命令を１個ずつフェッチおよびデコードし、次の命令に進む前に各命令の実行を完了することである。インテルＩ９６０プロセッサチップを例として用いると、ＩＳＳは、キャッシュ動作の十分なエミュレーションを提供する。これは、命令キャッシュおよびデータキャッシュの両方をサポートし、メモリコントローラレジスタ（mcon0～mcon15）を用いて、データをキャッシュするかどうかを決定する。また、このインタフェースは、メモリマップドハードウェアデバイスにアクセスするためのアドレス空間デコーディングも提供する。ルーチンは、ハードウェアルーチンのうちの１つ（例えば、uart_fetch）を呼び出すか、または、直接にＲＡＭエミュレーションにアクセスする。

【０１０２】ＩＳＳは、軽量オペレーティングシステムの下で組込みアプリケーションとして動作する例示的な"hello world"プログラムを実行するために用いられた。この簡単なプログラムは、４個のプロセス、すなわち、ＵＡＲＴドライバ、コンソールプロセス、'hello'アプリケーション、およびアイドルプロセスを含む。また、このプログラムは、オペレーティングシステムのコアと、Ｃランタイムライブラリも含む。

【０１０３】ターゲットシステムは、開始エントリポイントから開始される。ターゲットシステムは、ブランク記憶領域をゼロにクリアし、制御されたプロセッサリセットを実行して、内部制御レジスタを設定する。初期化ルーチンが、４個のプロセスを作成し、ＵＡＲＴのためのハードウェア初期化ルーチンを呼び出す。このハードウェア初期化ルーチンは割込みハンドラをインストールする。

【０１０４】コンソールプロセスは、ＵＡＲＴプロセスへのメッセージチャネルを開き、ＵＡＲＴキーボードインタフェースからの入力を受け取ることができるようにメッセージのキューをセットアップする。

【０１０５】アプリケーションは、Ｃランタイムライブラリを用いて、通常のＩ／Ｏファイル（stdin、stdoutおよびstderr）をコンソールマルチプレクサプロセスに対して開いた後、printfルーチンを呼び出して、文字列をフォーマットしてメッセージをコンソールに送る。これは、コンソールへのコンテクストスイッチを引き起こし、文字列の前にプロセス名を付加してそれをＵＡＲＴプロセスに転送する。

【０１０６】ＵＡＲＴは、ハードウェアによって生成されるTx-Available割込みに応答して、文字列を一時に１文字ずつハードウェアに送信する。すべての文字が送られた後、メッセージがコンソールプロセスを通じてアプリケーションに返され、その後、アプリケーションは終了する。

【０１０７】ＩＳＳをこのレベルで動作させると、ソフトウェアが正しく動作しているというある程度の信頼性

を示すことができる。例えば、アプリケーションとのＵＡＲＴの動作を制御するソフトウェアは、いつＣＰＵが文字を書くことができるかを示すためにチップによって供給されるステータスフラグを正しく解釈している。さらに、ルーチンは、正しいパラメータおよび連繋で、正しい順序で呼び出されている。また、検出可能なメモリ破損がなく、定義されたメモリ領域の外部へのアクセスもないと仮定することも妥当である。

【０１０８】しかし、このモードでシミュレータを動作させても、実行時間に関する実際の情報は得られない。あらゆる命令は１クロックティックしかかからないと仮定しているからである。命令タイミングモードを動作させ、パイプライントレースをプリントアウトすれば、命令サイクルがどこを進んでいるかが示される。

【０１０９】また、ＩＳＳを使用すると、回路内エミュレータを使用しても通常は入手できない情報にアクセスすることもできる。例えば、実際に使用されたレジスタキャッシュの最大深さを決定することや、キャッシュヒット統計とともにフレームこぼれ数をカウントすることも可能である。

【０１１０】また、それぞれの命令が実行された回数をカウントすることも可能である。このようなデータは、特に高集積ＳＯＣを製造するとき、プロセッサトレードオフを決定する際に非常に有益である。これは、例えば、より少数の分岐命令によって、ゲート数を少なくするためである。Ｉ９６０アーキテクチャは「予測分岐」命令を備えているが、コンパイラはそれらを使用しない。これらの命令を除去することも可能であり、あるいは、これらの機能を利用可能な別のコンパイラを検討することも可能である。

【０１１１】このレベルの詳細さでも、サイクル精度にはほど遠い。この場合、すべてのバストランザクションが（マルチワードオペレーションを含めて）同じサイクル内で完了するという１サイクルメモリモデルを仮定しているからである。このレベルでのタイミングを正確にモデル化するためには、システムデータバスとそのデバイスの精密なモデルが必要であり、これは、完全なコシミュレーション環境を生成することを必要とする。

【０１１２】次に、完全なサイクル精度の命令セットシミュレーションの構成について説明する。真のハードウェア／ソフトウェアコシミュレーションを可能にするためには、本発明の第１の特徴に要求されるＩＳＳの機能を少なくとも含む、選択されたＣＰＵに対する命令セットシミュレータが要求される。

【０１１３】しかし、この機能は、特定のＣＰＵを選択する際に次の３つの主要な要素を無視している。

【０１１４】１．例えば乗算や除算のように、多くの命令は、単一サイクルで実行を完了しない。

【０１１５】２．ＣＰＵは、リソース使用衝突がないと仮定して、バスオペレーションがレジスタ間オペレーションと同時に実行可能なように、複数の専用ユニットにわたる内部並列処理をインプリメントしていることがある。

【０１１６】３．内部キャッシュメモリ、外部ＲＡＭメモリおよびデバイスへのアクセスは、互いに非常に異なる応答時間を有することがある。これによる通常の効果は単に実行が遅延されるだけであるが、ある重大な場合には（例えば、到来する割込みの相対的タイミングがあるタスクの処理順序を決定する場合）、全体のパフォーマンスにおいて多大な役割を演ずることがある。

【０１１７】次に、電子回路とその電子回路を制御する制御プログラムとを有するシステムの検証のための、ターゲットプロセッサのサイクル精度の命令セットシミュレーションを導出する方法について詳細に説明する。

【０１１８】インテルＩ９６０プロセッサを例として用いると、ＩＳＳは、命令キャッシュに加えて、次の機能をインプリメントすることによって、Ｉ９６０命令スケジューラをモデル化する。

・並列デコーディングを有する４ワード命令パイプ。
・５個のＩ９６０命令パスのエミュレーション。
・結果が利用可能になる前には使用されないことを保証するためのレジスタスコアボーディング。
・分岐や呼出し（コール）のようなパイプライン切断命令の実装。
・完了するのに単一クロック時間より長くかかる命令に対するパイプラインストール。

【０１１９】図６を参照すると、命令パイプラインは、各プロセッサクロックティックの最初に検査される。パイプが完全に満たされてはいない場合、満たすためにifetchルーチンが呼び出される。パイプを満たすのに必要な命令が命令キャッシュに見つからない場合、ifetch擬似命令がスケジューリングされる。この命令は、ＣＰＵのバス制御ユニット（ＢＣＵ：Bus Control Unit）およびメモリパスにおいて実行される（そのため、これらのユニットがビジーである場合には遅延されることがある）。結果的に、この命令が実行されると、８ワードが命令キャッシュラインにロードされる。これらのワードのうちの４個までが、fill_Ipipeルーチンによって、命令パイプラインを満たすために使用される。使用されるワード数は、パイプの状態と、命令キャッシュバウンダリに関する現在の命令ポインタのアラインメントとに依存する。

【０１２０】命令パイプラインにデータがある場合、その値が命令へとデコードされる。一部のＩ９６０命令は複数ワードを占めるため、不完全な命令がパイプに存在する可能性があり、その場合（図６の第２のcallx命令の場合のように）、それはデコードすることができない。命令は一度だけでコードされ、デコードされた形がデータ構造体に格納される。また、デコーディングには、命令をスケジューリングするために必要な内部処理

ユニット、パイプラインおよびレジスタリソースを識別することも関連する。

【０１２１】ＣＰＵが使用中のリソースは、ティックごとに、スコアボード構造体によって表される。命令は、パイプラインから順に取り出され、１つずつスケジューリングされる。スケジューリングは、命令がこのティックの間にすでにコミットされたリソースを要求するとき、もしくは、命令がパイプライン切断を生じるとき、または、パイプライン内にもはやデコードされた命令がないときのいずれかに、停止する。

【０１２２】スケジューラがパイプライン内のスロットを消費した場合、残りのデータが繰り上がり、命令ポインタが更新される。

【０１２３】スケジューリングが完了すると、現在のティックにおいてユニットに入る命令が実行される。完了するのに単一ティックより長くかかる命令は、パイプラインがストールすると、そのリソースを後続のティックに伝搬させる。

【０１２４】図６を参照すると、ＡＧＵ、ＭＥＭバス、ソースレジスタとしてのフレームポインタ、および、デスティネーションレジスタとしてのスタックポインタをロックして、lda命令が現在のティックにおいてスケジューリングされることが可能である。これは、次のcallx命令がスケジューリングされないようにするため、次のcallx命令はパイプにとどまる。lda命令は、実行するのに複数サイクルかかるため、ＡＧＵは次のティックでストールし、この命令からのリソースは、この命令がＡＧＵを占有する限り、スコアボードにコピーされる。ldaは、完了すると、そのリソースを解放し、callx命令が開始することができる。次のcallx（これはここで完全にデコードされることが可能となる）は、次の２つの理由で、スケジューリングされない。第１に、このcallxでのリソースクラッシュのためであるが、それだけでなく第２に、callx命令はパイプライン切断（および新しい命令ポインタへの転送）を引き起こすからである。

【０１２５】図６を参照すると、lda命令のエミュレーションは特に複雑ではない。命令デコーディングルーチンは、デスティネーションレジスタおよびメモリオペランドをすでに評価しているため、ルーチンは、レジスタを保持している（ＩＳＳメモリ内の）ロケーションを単に更新するだけである。その結果は、検査のために外部観察者に対して表示することが可能であり、スコアボードは、この命令に対する全部で４ティックについてマークされる。

【０１２６】エミュレーションにおいては、デスティネーションレジスタが命令の期間中にアクセス不能とマークされている限り、現実のＣＰＵの場合のように「最終」ティックまで結果の代入を遅延させる必要はない。この便法は、ハードウェアがコシミュレートされているときにはもはや成り立たない。



【０１２７】図７～図９を参照して、サイクル精度のターゲットプロセッサのモデル化のプロセスについて説明する。Ｓ１１０５で、ターゲットプロセッサがオンボードキャッシュメモリを有するかどうかを判定する。ターゲットプロセッサがオンボードキャッシュメモリを有する場合、プロセスフローはＳ１１１５に進み、オンボードキャッシュアクセスを制御するアルゴリズムがモデル化される。ターゲットプロセッサがオンボードキャッシュメモリを有しない場合、プロセスフローはＳ１１２０に進む。

【０１２８】Ｓ１１２０で、ターゲットプロセッサが命令パイプラインアーキテクチャを有するかどうかを判定する。ターゲットプロセッサが命令パイプラインアーキテクチャを有する場合、Ｓ１１３０で、命令パイプライン充填プロセスがモデル化される。そうでない場合、プロセスフローはＳ１１３５に進む。

【０１２９】Ｓ１１３５で、命令およびリソースの表現を作成する。ターゲットプロセッサのアーキテクチャに依存して、各命令の効果、その命令が要求しロックするターゲットプロセッサ内のリソース、および、その命令の継続時間または終了条件を表現するデータ構造体が定義され初期化される。

【０１３０】図８を参照すると、Ｓ１１４０で、上記で定義されたリソース定義と、現在のマシン状態における命令のアクションとに対するインタプリタを作成することによって、命令がシミュレートされる。

【０１３１】Ｓ１１４５で、ターゲットプロセッサがマルチパスアーキテクチャを有するかどうかを判定する。ターゲットプロセッサがマルチパスアーキテクチャを有しない場合、プロセスフローはＳ１１６５に進む。これに対して、ターゲットプロセッサがマルチパスアーキテクチャを有する場合、Ｓ１１５５で、現在のマシン状態において、他の命令が並列に実行されるべきかどうかを判定するアルゴリズムが生成される。他の命令が並列に実行されるべきである場合、プロセスフローはＳ１１４０に戻り、並列命令が実行される。そうでない場合、プロセスフローはＳ１１６５に進む。

【０１３２】図８および図９を参照すると、Ｓ１１６５で、シミュレートされる命令がマルチサイクル命令であるかどうかを判定する。シミュレートされる命令がマルチサイクル命令である場合、Ｓ１１７０で、上記のようにスコアボードが用意される。次に、Ｓ１１７５で、上記のようにリソースが伝搬され、プロセスフローはＳ１１８０に進む。

【０１３３】Ｓ１１８０で、非決定性タイミングが存在するかどうかを判定する。非決定性タイミングが存在する場合、終了の評価を行う。Ｓ１１９０で、現在のマシン状態における非決定性タイミングオペレーションに対する終了条件を判定するアルゴリズムを生成する。ターゲットプロセッサのアーキテクチャに従って、これは、

エミュレートされるマシンレジスタ内の値に基づく計算を含むこともあり、また、外部信号の状態を調べるコードを生成することを含むこともある。Ｓ１１９２で、終了条件がチェックされ、命令が終了していない場合、プロセスフローはＳ１１７５に進む。命令が終了している場合、前に伝搬されたリソースがクリアされ、命令のシミュレーションは完了する。

【０１３４】図１０～図１３を参照して、電子回路と、その電子回路を制御する制御プログラムとを含むシステムのコバリデーションの実行について説明する。Ｓ１３１０で、刺激が試験対象システムに入力される。Ｓ１３１５で、ＣＰＵが「動作可能」であるかどうかを判定する。動作可能である場合、プロセスフローはＳ１３４０に進む。動作可能でない場合、Ｓ１３２０で、ターゲットプロセッサはストールされ、Ｓ１３２５で、エミュレートされているキャッシュメモリに命令がロードされる。Ｓ１３３０で、キャッシュメモリがロードされているかを判定する。メモリが十分にロードされている場合、Ｓ１３３５で、ターゲットプロセッサがアンストール（ストール解除）される。

【０１３５】Ｓ１３４０で、命令パイプラインが、ターゲットプロセッサの内部データフローモデルから導出される信号のシーケンスを用いて充填される。図１１を参照すると、Ｓ１３４５で、ターゲットプロセッサの内部データバスのアクションをエミュレートするために、所定数のクロックサイクルに基づく遅延が実行される。この所定数は、内部データバスの幅に基づき、１～数クロックサイクルの範囲とすることが可能である。

【０１３６】次に、Ｓ１３５０で、実行に利用可能な命令を解釈するために、ターゲットプロセッサの内部デコードサイクルが実行される。この命令デコードサイクルは、上記のように実行される。

【０１３７】ターゲットプロセッサの内部デコードサイクルが実行された後、Ｓ１３５５で、スケジューリングされた命令が上記のようにパイプライン切断をインプリメントしているかどうかを判定する。パイプライン切断が要求される場合、Ｓ１３６５で、命令スケジューラがストールされる。パイプライン切断が要求されない場合、プロセス制御フローはＳ１３７０に進む。

【０１３８】次に、Ｓ１３２０で、スケジューリングされた命令が、適当な命令パイプラインあるいはハードウェアコンポーネントに転送される。適当な命令パイプラインへの命令の転送後、Ｓ１３７５で、各命令のサイクル実行時間が計算される。

【０１３９】図１２を参照すると、Ｓ１３８０で、非決定性タイミングが存在するかどうかを判定する。非決定性タイミングが存在する場合、Ｓ１３９０で、Ｓ１１９０で生成された終了条件を判定するアルゴリズムが、各クロックサイクルの終端で実行される。

【０１４０】Ｓ１４００で、エミュレーションは、計算された命令時間に対するリソース割当てを伝搬させる。Ｓ１４０５で、計算された実行サイクルの終端で利用可能な結果が出力される。この結果は、エミュレートされたレジスタやフラグの値を更新することや、外部信号の値を変化させることを含むことがある。

【０１４１】次に、Ｓ１４１０で、次の命令サイクルで割込みハンドラがスケジューリングされるべきかどうかを判定する。スケジューリングされるべきである場合、Ｓ１４２０で、次の命令サイクルで実行するために割込みハンドラがスケジューリングされる。

【０１４２】図１３を参照すると、Ｓ１４２５で、刺激からの結果が試験対象システムから出力される。この出力は、外部観察者に対して示されるシミュレーションの進行の視覚的表示であることも可能であり、あるいは、後の検討や後処理のためのそのような変化のログ（例えば、図２０によって例示されるようなトレース出力）の生成であることも可能である。Ｓ１４３０で、試験対象システムに入力されるべき刺激がまだあるかどうかを判定し、もうない場合、プロセスは終了する。まだある場合、プロセス制御はＳ１３１０に進む。

【０１４３】次に、試験対象システムへの刺激の入力により、試験対象システムに結果を出力させることについて詳細に説明する。

【０１４４】一般に、いくつかの異なるプロセスが、試験対象システムの異なる部分をシミュレートする。これらのプロセスは試験対象システムに固有であり、ケースバイケースで修正されなければならない。図１４を参照すると、設計された試験サイクルを継続的に繰り返して、コシミュレートされるハードウェアおよびソフトウェアを十分に作用させるために、例として、グローバルループコントローラ２３が用いられる。試験刺激プロセス２４は、「外界」から試験対象システムに刺激を提供する。試験刺激プロセス２４のもう１つの機能は、誤った入力がどのように処理されるかを見るために、誤入力を試験対象システムに入力することである。例えば、試験刺激プロセス２４は、データ転送トランザクションの応答側として作用し、試験対象システムの再試行メカニズムを試験するために、プロトコルエラーで応答することが可能である。バスアービトレーションプロセス２２は、試験対象システムと試験刺激プロセスの間のバスアービトレーションを提供する。試験対象回路２１は、単一のプロセスであることも可能であり、あるいは代替例では、回路設計の複雑さに依存して、数個のプロセスであることも可能である。

【０１４５】さらに、信号表示プロセスにより、ステップ実行を行い、グローバル状態に対する個々のステートメントの効果をウォッチすることが可能である。

【０１４６】次に、電子回路のハードウェアモデルを命令セットシミュレーションに結合して、軽量コンピュータオペレーティングシステム環境で実行される試験対象

システムを作成することについて、さらに詳細に説明する。

【０１４７】第１段階として、ハードウェアの動作記述を調べ、各ハードウェアプロセスが個別のソフトウェアプロセスによって表現されるプロセス構造が構成される。各プロセス内では、動作記述が、軽量オペレーティングシステムのプログラミング言語に翻訳される。命令セットシミュレータがこの言語で同様にコーディングされる。シミュレータに必要なファブリックプロセスおよび刺激生成プログラムも同様である。第２段階として、プログラミング言語ソースファイルは、軽量オペレーティングシステムが動作するプロセッサに対するオブジェクトコードにコンパイルされる。第３段階として、これらのファイルが、軽量オペレーティングシステムに対するオブジェクトコードとリンクされ、完全な実行環境が形成される。第４段階として、シミュレータが実行されるシステムに、実行可能ファイルがロードされる。第５段階として、シミュレータが、軽量オペレーティングシステムの制御下で実行され、出力が観察される。

【０１４８】次に、図１４を参照して、電子回路と、その電子回路をターゲットとする制御プログラムとのコバリデーションのためのコンピュータシステムについて、簡単に説明する。電子回路および制御プログラムは、軽量コンピュータシステムオペレーティング環境で実行される所定のコンピュータ言語を用いてシミュレートされる。また、このコンピュータシステムは、上記のルーチンを使用する。Ｓ０９００の設計ステップの終了後、コンピュータシステムは、ファブリックプロセス１０を有する。ファブリックプロセス１０は、複数のグローバル信号の状態を管理する信号メンテナサブシステム１２と、ソフトウェアモデルおよび命令セットシミュレータによって使用されるクロック信号を生成するクロックジェネレータサブシステム１３と、ソフトウェアモデルおよび命令セットシミュレータからのイベントをウェイトするプロセスキューを管理するキューメンテナサブシステム１４と、所定のタイミング間隔で少なくとも１つのプロセスを実行するスケジューラサブシステム１６と、少なくとも信号メンテナサブシステム１２、クロック信号ジェネレータサブシステム１３、キューメンテナサブシステム１４およびスケジューラサブシステムの実行を制御する中央制御プロセス１１とを有する。

【０１４９】マイクロカーネルサポートプロセス以外に、コンピュータシステムは、ファブリックプロセス１０、命令セットシミュレーションプロセス、ならびに、ハードウェアコンポーネントおよび刺激システムを表すプロセスを有し、これらはすべて互いに関連して動作する。コシミュレーション環境の中央制御プロセス１１は、次の５個の主要なサブシステムの全体の機能および相互作用を管理する。

（ａ）すべてのグローバル信号の状態を管理する信号メンテナ１２。

（ｂ）クロック信号を生成するクロックジェネレータ１３。

（ｃ）イベントをウェイトするプロセスのキューを管理するキューメンテナ１４。

（ｄ）各ナノ秒ティックごとにそれぞれのアクティブプロセスを実行するスケジューラ１６。

（ｅ）ユーザに対して信号変化を示すディスプレイを駆動するディスプレイジェネレータ１５。

試験刺激の例としては、外部インタフェース上でのデータ利用可能性を示す信号がある。また、刺激は、エラー条件、例えば、ＥＣＣメモリによって生成されるパリティ検査エラーをシミュレートすることも可能である。

【０１５０】図１５を参照すると、シミュレーション環境の全体は、軽量オペレーティングシステム３０内の複数のプロセスとして動作する。個々のプロセスには、ファブリックプロセス３１、命令セットシミュレーションプロセス３２およびＶＨＤＬプロセス３３、３４が含まれる。ＶＨＤＬプロセス３３、３４は、モデル化されているハードウェアのインスタンス化されたハードウェアプロセスと一対一に対応する。すべての信号相互作用３５は、オペレーティングシステム３０によってスケジューリングされたメッセージとして、ファブリックプロセス３１を通る。

【０１５１】このようなソフトウェアベースのシミュレーションシステム内では、シングルプロセッサシステムでは真の並行性はないため、見かけ上の並列処理しか達成することができない。これは、信号に対する「同時」変化と時間の概念の問題を引き起こす。考慮すべき次の３つの問題がある。

【０１５２】・問題１：　２つのプロセスが動作可能であり、それらの実行の期間中に、ある信号の値を変化させる場合、その結果は、プロセスが実行される順序に依存することがある。例えば、信号"test"が実行の最初に値'1'を有する場合、

プロセス１：　if（test = '1'）then test <= '0'; end if;

プロセス２：　test <= '1';

【０１５３】プロセス１がプロセス２の前に実行される場合、（素朴な）結果は、実行の終了時に"test"が'1'にセットされるというものである。プロセス２がプロセス１の前に実行される場合、結果は、実行の終了時に"test"が'0'にセットされるというものである。

【０１５４】・問題２：　プロセスが変化し、同じ実行期間に、ある信号を読み出す場合、結果は非決定性となる。例えば、次のコードフラグメントを考える。

プロセス１：　test <= '1';

if（test = '1'）then output <= '1'; end if;

現実のシステムでは、これはoutputを'1'にセットすることもしないこともある。

【０１５５】・問題３： プロセスは、既知の周波数で動作するクロック信号に対する絶対時間の値を必要とする。さらに、絶対時間は、より複雑なデューティサイクルとともに、次のような構文に必要である。

クロックプロセス： clk <= !clk after 5 ns;

【０１５６】従来のＶＨＤＬシミュレータは通常、最初の２つの問題を、「デルタ」、すなわち、サブティック同期ポイントの概念により処理する。サブティック同期ポイントを使用すると、すべての信号値は、デルタの最初に固定される。これらの値が信号値に対する要求に応答して出され、デルタの最後にのみ値が更新される。これは、シミュレーションを決定性にし、プロセスのシミュレーションの順序とは独立にするという利点がある。これは、ＶＨＤＬプログラマに対して、次のデルタへのアラインメントを強制するように余分のウェイト(wait)ステートメントをコードに挿入することを要求する。上記の問題２の場合、この非決定性問題を解決するために、デルタバウンダリを強制するステートメントが挿入される。

プロセス１： test <= '1';
wait for 0 ns;
if (test = '1') then output <= '1'; end if;

プロセス１は一貫してoutputを'1'にセットすることになる。しかし、合成された結果が期待通りに動作するという保証はない。

【０１５７】ファブリックプロセスの現在の実装では、瞬時値が出される（そのため、この例は、ウェイトが暗黙のうちに挿入されているかのように動作する）が、結果は、プロセスの実行の順序に依存する。当業者には明らかなように、システムは、１つの信号に対する単一の「デルタ」内の複数の変化を検出して、これをシミュレーション出力に報告するか、あるいは、他の動作をまねるように実装を変更するように、修正することも可能である。

【０１５８】シミュレータは、絶対時間の値を生成する。ファブリックプロセスのクロックサブシステムは、現在、１ナノ秒の基本タイムスライスを使用しているが、これは、現在のクロック信号の必要な精度を追跡するのに十分である。これは、数ＧＨｚのシステムがさらに広まるに応じて、変更される必要があるかもしれない。クロック信号のサポートは、共通のオペレーションのためのインタフェースを単純化するために、中央プロセスに組み込まれた。

【０１５９】ほとんどの動作ＶＨＤＬは、２つのモードのうちの一方で動作するトリガで書かれる。非同期モードでは、信号変化は、その信号に反応するプロセスを、次のナノ秒で動作可能にした。同期（クロック）モードでは、信号は、クロック信号の端（立上りあるいは立下り）でのみ調べられた。これらのモードは、図２のＳ１０００における動作翻訳中に決定される。プロセスの動



作可能性は、前のティックに生じた信号変化と、そのときに起こっているクロック遷移とに基づいて、各ナノ秒ティックの最初に決定された。そのナノ秒ティック中に動作可能となったプロセスは次のティックまで開始されず、「デルタ」メカニズムに類似の環境提供した。

【０１６０】このインタフェース方法の顕著な特徴は、イベントをウェイトしているプロセスが、マイクロカーネルコア内で真のウェイト状態にあることである。すなわち、それらのプロセスは、各クロックティックでチェックされず、システムの残りの部分におけるオーバーヘッドを生じない。ファブリックプロセスは、条件のすべてのチェックを処理する。従来のオペレーティングシステムのオーバーヘッドを除去し、軽量オペレーティングシステムの直下でシミュレーションを実行することによって、本発明は、より高速に動作するのみならず、エミュレートされる状態の表現は、オペレーティングシステム自体のスケジューリングコアを通じて直接にインプリメントされる。

【０１６１】個々のプロセスが翻訳されたコードを実行するため、トレース出力はコンソールに表示することができる。システム全体のECHOVHDL変数が０でない場合、シミュレーション環境は、ナノ秒ティックのタイムスタンプを前に付した文字列を印字することによって信号値の変化が観察者に見えるようにする。コシミュレーション環境内の信号表示サブシステムと組み合わせて、ステップ実行を行い、グローバル状態に対する個々のＶＨＤＬステートメントの効果をウォッチすることが可能である。

【０１６２】図１６に、プロセッサ４０、Ｉ／Ｏデバイス４３およびビデオディスプレイ端末４１を有するコンピュータシステムの実施例を示す。Ｉ／Ｏデバイス４３は、キーボードおよびマウスを含むが、これらに限定されない。タッチパッドのような他のデバイスも使用可能である。さらに、コンピュータシステムは、このコンピュータシステムが本発明のステップを実行することを可能にするように適応したソフトウェア命令を含むメモリ４２（図示していないが、プロセッサ４０に組み込まれている）を有する。

【０１６３】また、コンピュータシステムは、データリンク４４によってプロセッサ４０に接続されたサーバ４５も含むことが可能である。データリンク４４は、従来のデータリンク（例えば、イーサーネット、ツイストペア、ＦＴＰ、ＨＴＴＰなど）である。サーバ４５は、このサーバに接続されたプログラムライブラリ４６へのアクセスを提供する。また、プログラムライブラリ４６は、コンピュータシステムが本発明のステップを実行することを可能にするように適応したソフトウェア命令を提供することも可能である。上記のように、プログラムライブラリ４６は、当業者に周知の任意のさまざまな媒体（例えば、フロッピー（登録商標）ディスク、ハード

ディスク、光ディスク、カートリッジ、テープ、ＣＤ－ＲＯＭ、書き込み可能ＣＤなど）上に実現可能である。図１６に示したコンピュータシステムでは、メモリ４２上のソフトウェア命令により、プロセッサ４０は、データリンク４４を通じてサーバ４５にアクセスすることによって、プログラムライブラリ４６にアクセスすることができる。図１６に示したコンピュータシステムは、いかなる意味でも限定的であることは意図しておらず、本発明を実施する多数のさまざまなコンピュータシステムを組み合わせることが可能である。

【０１６４】図１７に、本発明によってモデル化される例示的なシステムのブロックレベル設計を示す。システムは、Ｉ９６０プロセッサ３５、グルーロジック３６およびＵＡＲＴ３７を有する。この例示的システムは、本発明の適用範囲を制限するものではなく、以下の説明に役立てるためのものである。このサンプルシステムは、外部データバスがメモリサブシステム３８およびアドレスデコーディンググルーロジック３６の両方に接続されたＩ９６０命令セットシミュレータを有する。矢印は、信号の一方向性または双方向性を示す。グルーロジック３６は、メモリおよびＵＡＲＴチップ３７に対するchip-select信号をドライブするとともに、ＵＡＲＴ３７に対してさまざまな信号提示要求を提供する。この設計は、プロセッサクロックおよびバスクロックという２つのクロックにより、意図的に複数のタイミングドメインに分割された。

【０１６５】コシミュレーション環境を実証するため、Ｉ９６０命令セットシミュレータは、ＮＥＣ　ＶＲ４３００プロセッサのユーザガイド(User Guide)からの情報を用いて、ＭＩＰＳアーキテクチャに基づくデータバスにリンクされた。バス上には外部バスマスタリングデバイスがなかったため、システムは、フルＭＩＰＳバスのサブセットを使用した。選択された信号のセットは、さまざまなバースト能力とともに、さまざまな応答時間で、デバイスをインプリメントするのに十分であった。

【０１６６】バスは同期モードで動作し、アクティビティはbusclk信号の立上り端でスケジューリングされる。信号およびプロトコルは以下に記載するとおりである。次のバス信号が用いられる。

・SysAD [31..0] 多重化システムアドレスおよびデータバス。

・SysCmd [4..0] データおよびトランザクションタイプを識別するコマンドバス。これは、リードモードに８ワードのバーストを追加するが、エラー指示コマンドおよび外部非応答モードを使用しない。

・EOK 外部デバイスがコマンドを受け入れることができるときにローにセットされる。

・EValid 外部デバイスがバス上に有効なデータを置いたときにローにセットされる。

・PMaster プロセッサが（外部デバイスがリード応答を発行することができるように）バス所有権を解放したときにハイにセットされる。外部バスマスタリングデバイスのないこの簡単なモデルでは、フルＭＩＰＳバスからの他のプロトコル信号は使用されなかった。

【０１６７】プロセッサの視点から、図１８を参照すると、バスは次の順序でドライブされる。

１．プロセッサは、EOKがローになる（これは、バスを使用中の外部デバイスがないことを示す）のをウェイトする。

２．プロセッサは、SysAD上にアドレスを、また、SysCmd上に適当なWriteコマンドをドライブする。また、プロセッサは、PValidをローにセットする（これは、バス上に有効なデータがあることを示す）。

３．プロセッサは、EOKがハイになる（これは、外部デバイスがコマンドを受け入れたことを示す）のをウェイトする。

４．プロセッサは、SysADバスに、busclkサイクルごとに１個ずつデータワードを順次ドライブし、最後（または唯一）のワードを'no-more'フラグでマークする。

５．最後のデータサイクルの終わりに、プロセッサは、PValidをハイにセットし（これは、プロセッサがもはやバスをドライブしていないことを示す）、SysADおよびSysCmdをトライステートにする。

【０１６８】図１９を参照すると、プロセッサリードは次のようになる。

１．プロセッサは、EOKがローになる（これは、バスを使用中の外部デバイスがないことを示す）のをウェイトする。

２．プロセッサは、SysAD上にアドレスを、また、SysCmd上に適当なReadコマンドをドライブする。また、プロセッサは、PValidをローにセットする（これは、バス上に有効なデータがあることを示す）。

３．プロセッサは、EOKがハイになる（これは、外部デバイスがコマンドを受け入れたことを示す）のをウェイトする。

４．プロセッサは、PValidをハイにセットし（これは、プロセッサがもはやバスをドライブしていないことを示す）、PMasterをハイにセットしてバスの制御を解放する。

５．プロセッサは、EValidがローにセットされる（これは、外部デバイスがバス上に有効なデータを置いたことを示す）のをウェイトする。次に、プロセッサは、SysADバスから、busclkサイクルごとに１個ずつデータワードを順次読み出す。

６．プロセッサは、EValidがハイにセットされるのをウェイトしてから、PMasterをローにセットし、バスの制御を再び獲得する。

【０１６９】ＵＡＲＴは、次のような非常に単純な信号モデルを使用した。

・uartD [7..0] は、チップとの間での単一データ（非

39

バースト）アクセスをサポートするバイト幅双方向データバスである。

・uartAは、２つの内部レジスタのうちの一方を選択するために用いられる１ビット「アドレス」バスである。アドレス'0'は制御レジスタであり、アドレス'1'はデータレジスタである。

・uartRは、プロセッサがリード要求を発行した場合にローにセットされる。

・uartWは、プロセッサがライト要求を発行した場合にローにセットされる。

・uartSは、ＵＡＲＴを選択するためにローにセットされる。アドレスおよびデータバスの値は、チップが選択されている間中有効でなければならない。

・uartIは、ＵＡＲＴからプロセッサに割込みを通知するためにハイにセットされる。

【０１７０】busclkは、チップに対する同期クロック信号であり、すべての遷移は、クロックの立上り端でサンプリングされる。

【０１７１】ＵＡＲＴは、アドレスおよびデータの値が同時に提示されることを要求するため、グルーロジックは、多重化されたSysADバスからの翻訳を処理しなければならない。

【０１７２】

【表１】

テーブル２

| | | |
|---|---|---|
| ライト | 0x38レジスタ０ | リセット割込み |
| ライト | 0x02レジスタ１ | 割込み送信イネーブル |
| リード | 0x04レジスタ０ | バッファエンプティ送信 |

【０１７３】制御レジスタは、まずインデックス番号を書き込んだ後に８ビットのデータを書き込むという２段階アクセスによって、内部ＵＡＲＴレジスタにアクセスするために用いられる。制御レジスタ０は、制御アドレスに制御オペレーションを書き込むことによって直接にアクセスされることが可能である（制御オペレーションは、間接レジスタ番号の範囲外のバイナリ値を有する）。この最小モデルでは、上記テーブル２にリストした制御オペレーションのみがインプリメントされた。これらは、割込み駆動およびポーリングモードの出力（入力はない）を提供するのに十分であった。

40

【０１７４】システムメモリは、busclkの速度でSysADラインを通じてのバックツーバックバーストモード転送をサポートすると仮定された。メモリコントローラには、プロセッサバス上のSysAD、SysCmd、PMasterおよびEValid信号へのアクセスが許可され、busclkでクロックされた。メモリコントローラは、グルーロジックへのさらに２つの次の信号を使用した。

・memSはメモリchip-selectであり、これは、メモリが、バス上のコマンドおよびアドレスデータのターゲットであったときにローにセットされる。

・memEOKは、メモリコントローラがコマンドを受け入れることができるときに、メモリコントローラによってローにセットされる。

【０１７５】「現実の」コントローラの内部の詳細（例えば、ＲＡＳおよびＣＡＳ生成）はモデル化されず、すべてのメモリアクセスは単一の（バス）クロックサイクルで応答された。

【０１７６】ＵＡＲＴおよびメモリの実装はいずれも、上記のアーキテクチャ仕様によるそれらのオペレーションを制御するために、ＶＨＤＬ信号ライブラリを用いて直接にbehavioral Cで書かれた。さらに、ＵＡＲＴ実装は、ＩＳＳログファイルに出力されるように命令された文字列を格納した。このことは、シミュレータは、シミュレートされたシステムの出力の視覚的記録を生成したことを意味する。グルーロジックは、アドレスでコーディングおよび信号翻訳のための状態マシンを実装するＶＨＤＬで、単一のエンティティとして書かれた。ＶＨＤＬは、Ｓ１０００の方法に従ってＣに翻訳された。

【０１７７】次に、上記の命令セットシミュレータをコシミュレーション環境に統合することについて説明する。ＩＳＳをファブリック環境に統合するためには、ＩＳＳにいくつかの変更が必要とされた。いったんＩ／Ｏオペレーションが予測可能な継続時間のものでなくなると、その変更のほとんどは、タイミングの変更に関するものであった。

【０１７８】ファブリックインタフェース、プロセスの同期クロックおよび信号のセットを初期化するために、次のコードがプロセス初期化シーケンスに追加された。

【０１７９】

```
FabInit();
/* pclock周期を指定する */
fabric_set_clock("pclk", 0, 5);
/* プロセッサ信号の初期状態 */
Signal(PMaster, 0);
Signal(PValid, 1);
/* ドライブされるまでデータおよびコマンドバスは浮遊 */
SignalZ(SysAD);
```

```
SignalZ(SysCmd);
```

【０１８０】単一のプロセッサクロックティックをシミュレートするルーチンは、各ティックを実行する前に、

```
WaitClkR();
```

への呼出しを追加することによって、中央ファブリックプロセスからのpclk上の立上り端変化をウェイトするように変更された。これはまた、プロセッサを他のハードウェアコンポーネントと同期させる効果を有する。注意すべき点であるが、このオペレーションのシーケンスは、選択されたデータバスモデルのアーキテクチャによって決定されているため、モデル化されているシステムに固有である。このような統合ステップは、コシミュレートされるそれぞれの設計ごとに実行されることになる。

【０１８１】入出力（フェッチおよびストア）オペレーションの基本的なストラテジは同じままであった。メモリ領域デスクリプタに依存して、データキャッシュ（または命令キャッシュ）をまず参照した後、データがキャッシュにない場合には、外部デバイスのうちの１つがアクセスされる。主な変更は、すべてのアドレスデコーディングがグルーロジックで行われ、「ハードウェア」記述およびdevice_storeルーチンが使用されなかったことである。

【０１８２】インタフェース自体は、上記の説明に従った。例えば、新たなオペレーションは次のように開始された（変数thisはＩ／Ｏ要求データ構造体を指すと仮定する）。

【０１８３】

```
if (GetSignal(EOK) == 0)

{
    Signal(Pvalid, 0);
    Signal(SysAD, this->addr);
    Signal(SysCmd, this->op);
    this->memstate = 1;
}
sbmarksb(this->inst, 2);
```

すなわち、EOKがローにセットされるのをウェイトした後、バス上に信号をドライブする。なお、コマンドが開始されたかどうかにかかわらず、ユニットは次のティックの間ストールされ、リソースは利用可能でないとマークされなければならない。また、このルーチンはそのすべての信号に対してポーリングを行う。ウェイトポイントが'step'ルーチンにあるからである。

【０１８４】外部バスマスタリングデバイス、あるいは、エラー処理がない場合、外部インタフェースは、状態が１０個より少ない小規模な状態マシンで捕捉された。また、このマシンは、命令フェッチサイクルも処理した。

【０１８５】このようなシステムに割込み信号を統合する際には特別の注意をしなければならない。シングルプロセス実装では、シミュレートされるプロセッサの割込み処理中の制御レジスタを直接修正することが可能である。コシミュレーションシステムでは、これは、ＵＡＲＴコードにおける

```
Signal(uartI, 1);
```

への呼出しと、ＩＳＳプロセスにおける対応するテスト

```
if (GetSignal(uartI) == 1)
 {
    IGEN(7);
 }
```

によって置き換えなければならない。このアプローチは、バス制御ユニット(BusControl Unit)がビジーでありプロセッサが割込みを直ちに処理することができないとき、または、割込みが実行可能になる前に命令フェッチを要求するときに問題を引き起こした。特別の対処がなければ、プロセッサの優先度レジスタが割込み優先度に更新されるまで、割込みは各クロックティックにおいて処理されることになる。

【０１８６】これは、割込み処理が開始されるとセットされるフラグにより処理された。このフラグは、バス制御ユニットにおけるアクティビティ（割込みディスパッチャによってスケジューリングされるアクティビティを含む）がある限り、すべてのパイプラインスケジューリングをバイパスした。このフラグをセットすることは、割込み処理中のレジスタを検査するオペレーションが再びスケジューリングされる前にすべてのＩ／Ｏオペレーションが完了してプロセッサ優先度が更新されることを保証した。

【０１８７】プロセッサクロックは、１００ＭＨｚで動作するようにセットされた。プロセッサ内では、命令キャッシュと命令パイプの間の内部バスは、フルスピードで動作するフル１２８ビット幅と仮定された。したがって、命令パイプはキャッシュからレイテンシ０で充填される。データキャッシュは、レベル１、すなわち、フルプロセッサスピードにあると仮定された。メモリアクセスはbusclkで動作したため、アクセスあたり１個の（プロセッサ）ウェイト状態を必要とした。

【０１８８】ＵＡＲＴは、９６００ボーの外部シリアルラインに接続されていると仮定された。したがって、文字は１ミリ秒ごとに送出された。ＵＡＲＴ内部にはバッファリングがないため、'Tx Available'信号は、ライトオペレーションごとに１ミリ秒だけ遅延された。

【０１８９】このシステムは、前述のコード例、すなわち"Hello World"プログラムの実行を繰り返すために使用された。図２０に示す出力は、ＭＩＰＳ－ＩＩＩアーキテクチャＣＰＵを含む開発ボード上で実行された、動作中のシステムを実証している。このアーキテクチャは、シミュレートされているＣＰＵ（インテルＩ９６

０）のアーキテクチャとは異なり、より能力の高いプロセッサがどのようにシミュレーションのために用いられるかを実証している。

【０１９０】ターゲットＣＰＵ自体の処理能力が低い場合、ターゲットＣＰＵは、シミュレーション環境をサポートしておらず、適当な時間で結果を出さない可能性がある。軽量オペレーティングシステムのためのデバイスドライバを容易に書くことができるため、迅速なプロトタイピングとシミュレーション環境が提供可能である。シミュレータをシングルＣＰＵに制約する理由はない。

【０１９１】軽量オペレーティングシステムコア自体を除いて、シミュレーションシステムは、特定のマシンアーキテクチャとは独立である。その非常に低いオーバーヘッドおよびメモリ要求により、軽量オペレーティングシステムは、ページングやスワッピングによらずに、利用可能なＲＡＭをシミュレーションのために最大限に利用させることができる。また、このようなシステムは一般に、最小限の外部周辺装置のセットしか必要としないため、従来のワークステーションのためのシミュレーションエンジンあるいはアクセラレータとして高性能ＣＰＵの利用することも考えられる。

【０１９２】図１６を参照すると、本発明のもう１つの特徴によれば、目的とするターゲットＣＰＵは、通信媒体５１を通じてシミュレーションシステム４０に接続された開発ボード５０上に位置する。通信媒体５１は、システム４０内に共存するデータバスであることも可能である。ターゲットアプリケーションは、開発ボード５０のＣＰＵ上で直接実行され、命令セットシミュレータの必要性が回避される。このアプローチは、サイクルレベルの制度を維持するためにＣＰＵ命令セットからのサポートを必要とし、任意の与えられたＣＰＵに対するその実現は、すでに説明した原理から導出することが可能である。

【０１９３】本発明のさまざまな特徴についての以上の記載は、例示および説明の目的で提示したものである。これは、網羅的であることや、開示したとおりの形に本発明を限定することは意図しておらず、さまざまな修正および変更が、上記の説明に照らして可能であり、また、本発明の実施により得られる。当業者が、考えている個々の用途に合わせて、さまざまな実施例においてさまざまな修正を施して本発明を利用することができるように、本発明の原理およびその実際的な応用について説明した。

【０１９４】したがって、本発明のいくつかの特徴についてしか具体的には説明しなかったが、明らかなように、本発明の技術思想および技術的範囲から離れることなく、さまざまな変形を行うことが可能である。さらに、頭字語は、単に、本明細書の可読性を高めるために用いられている。注意すべき点であるが、このような頭字語は、用いられている用語の一般性を狭めることを意図しておらず、特許請求の範囲を限定すると解釈されてはならない。

【図面の簡単な説明】

【図１】マルチレイヤシミュレーション環境を有する従来のシミュレーションシステムを示す図である。

【図２】本発明の特徴による、電子回路と、その回路をターゲットとするソフトウェアとのコバリデーション方法の基本プロセスフローを示す図である。

【図３】本発明の特徴による、ハードウェアの動作記述をソフトウェアモデルに翻訳する方法の詳細なプロセスフローを示す図である。

【図４】本発明の特徴による、ハードウェアの動作記述をソフトウェアモデルに翻訳する方法の詳細なプロセスフローを示す図である。

【図５】命令セットシミュレータへのコマンドラインインタフェースの例示的なコマンドのセットを示す図である。

【図６】インテル１９６０プロセッサによる代表的な命令フローを示す図である。

【図７】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図８】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図９】本発明の特徴による、命令がターゲットマイクロプロセッサを通る際の命令のサイクリングを決定するプロセスフローを示す図である。

【図１０】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図１１】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図１２】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図１３】本発明の特徴による、軽量コンピュータオペレーティングシステム環境でコバリデーションシミュレーションが実行される場合の、エミュレートされるターゲットプロセッサのプロセスフローを示す図である。

【図１４】本発明の特徴による、軽量コンピュータシステム環境におけるコバリデーションの例示的なソフトウェアプロセスを示す図である。

【図１５】本発明の特徴による、軽量コンピュータオペレーティングシステム環境における信号フローパスを示す図である。

45

【図１６】本発明の特徴による、ターゲットマイクロプロセッサおよび電気回路のハードウェアおよびソフトウェアコバリデーションのための例示的なコンピュータシステムを示す図である。

【図１７】本発明を用いて協調検証されるターゲットマイクロプロセッサ、グルーロジック、Ｉ／Ｏハードウェアおよびメモリからなる例示的なシステムを示す図である。

【図１８】図１７に示した例示的システムのバスリードタイミングを示す図である。

【図１９】図１７に示した例示的システムのバスライトタイミングを示す図である。

【図２０】図１７に示した例示的システムに対して、本発明によって生成される信号タイミングを示す図である。

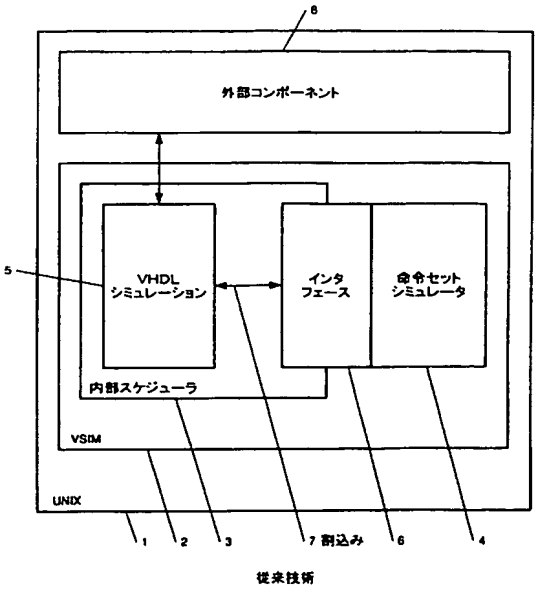
【符号の説明】

１　ＵＮＩＸシステム
２　ＶＳＩＭ環境
３　内部スケジューラ
４　命令セットシミュレータ
５　ＶＨＤＬシミュレーション
６　インタフェース
７　割込み
８　外部コンポーネント
１０　ファブリックプロセス
１１　中央制御プロセス
１２　信号メンテナサブシステム

46

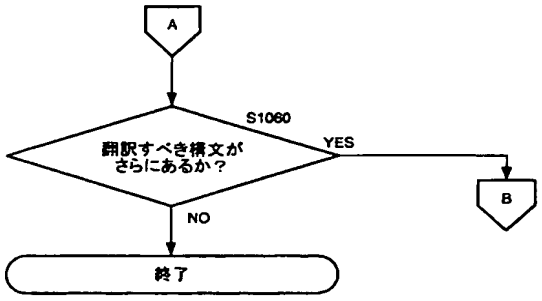
１３　クロックジェネレータサブシステム
１４　キューメンテナサブシステム
１５　ディスプレイジェネレータ
１６　スケジューラサブシステム
２１　試験対象回路
２２　バスアービトレーションプロセス
２３　グローバルループコントローラ
２４　試験刺激プロセス
３０　軽量オペレーティングシステム
３１　ファブリックプロセス
３２　命令セットシミュレーションプロセス
３３　ＶＨＤＬプロセス
３４　ＶＨＤＬプロセス
３５　Ｉ９６０プロセッサ
３６　グルーロジック
３７　ＵＡＲＴ
３８　メモリサブシステム
４０　プロセッサ
４１　ビデオディスプレイ端末
４２　メモリ
４３　Ｉ／Ｏデバイス
４４　データリンク
４５　サーバ
４６　プログラムライブラリ
５０　開発ボード
５１　通信媒体

【図１】

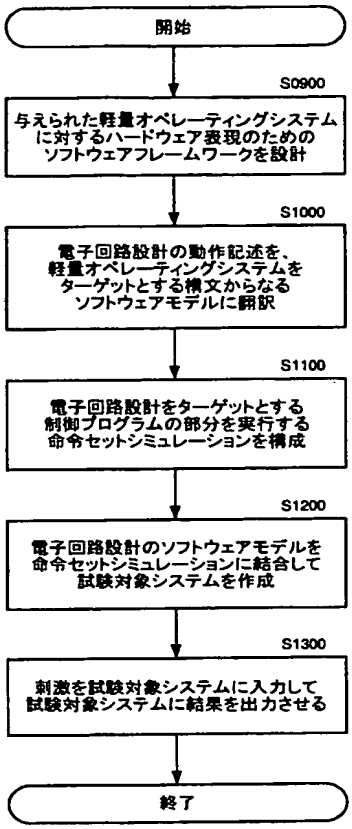



従来技術

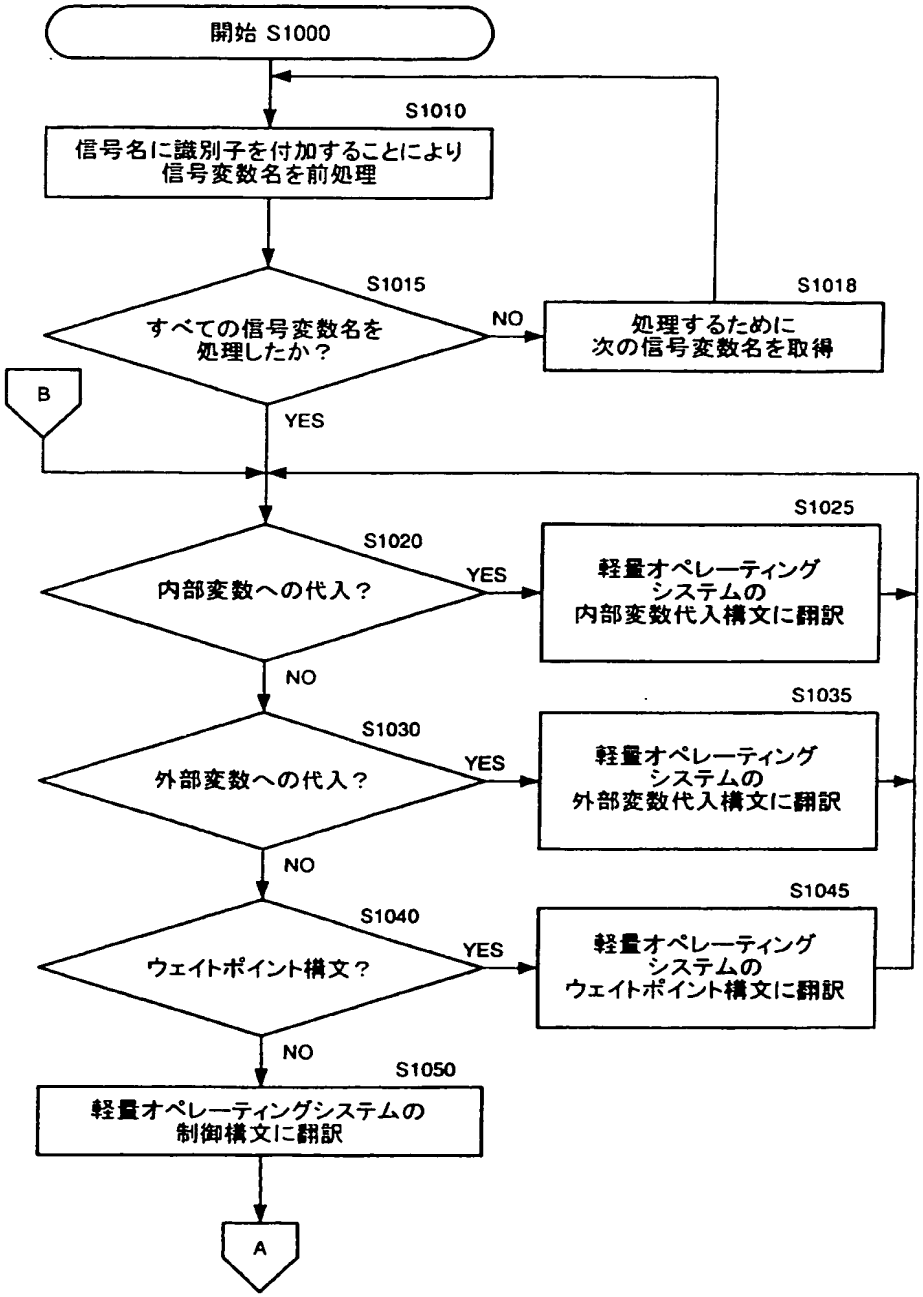
【図４】



【図２】


開始
S0900
与えられた軽量オペレーティングシステムに対するハードウェア表現のためのソフトウェアフレームワークを設計
S1000
電子回路設計の動作記述を、軽量オペレーティングシステムをターゲットとする構文からなるソフトウェアモデルに翻訳
S1100
電子回路設計をターゲットとする制御プログラムの部分を実行する命令セットシミュレーションを構成
S1200
電子回路設計のソフトウェアモデルを命令セットシミュレーションに結合して試験対象システムを作成
S1300
刺激を試験対象システムに入力して試験対象システムに結果を出力させる
終了

【図３】

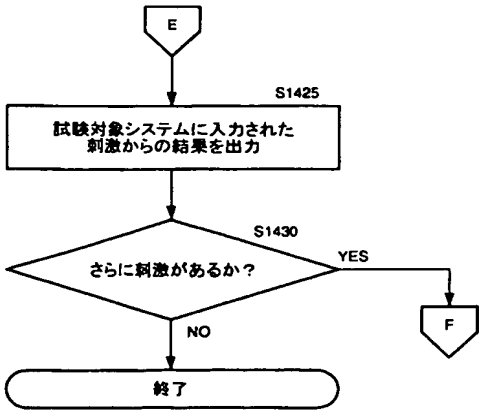

開始 S1000
S1010
信号名に識別子を付加することにより信号変数名を前処理
S1015
すべての信号変数名を処理したか？
NO
S1018
処理するために次の信号変数名を取得
B
YES
S1020
内部変数への代入？
YES
S1025
軽量オペレーティングシステムの内部変数代入構文に翻訳
NO
S1030
外部変数への代入？
YES
S1035
軽量オペレーティングシステムの外部変数代入構文に翻訳
NO
S1040
ウェイトポイント構文？
YES
S1045
軽量オペレーティングシステムのウェイトポイント構文に翻訳
NO
S1050
軽量オペレーティングシステムの制御構文に翻訳
A

【図５】

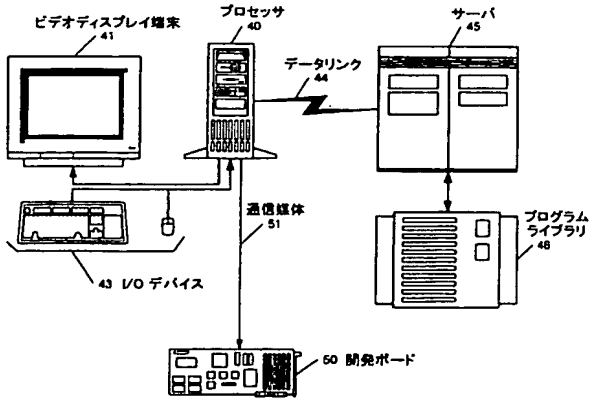
| コマンド | アクション |
|---|---|
| All | すべてのレジスタ／状態変数を表示する |
| Stats | 統計処理を実行する |
| ICounts | 命令ごとのカウンタ |
| Mem <a> <w> | アドレス<a>から<v>まで幅<w>でメモリを表示する |
| Set <r> <v> | レジスタ<r>を値<v>にセットする |
| Show <r> | レジスタ<r>の値を印字する |
| Step | シングルステップ１命令 |
| Go <n> | たかだか<n>個の命令だけステップする |
| Break <a> | アドレス<a>にブレークポイントをセットする |
| IStop [on\|off] | 割込みで停止する |
| Print [on\|off] | 命令ごとの印字をセットする |
| Time [on\|off] | 命令ごとのタイミングをセットする |
| Int <n> | 割込み<n>を発生する |
| XLoad <f> [B] | ターゲットファイル<f>をロードする（Bはビッグエンディアン） |
| Map <f> | ターゲットシンボルファイル<f>をロードする |
| Local <f> <a> <s> [Clear] | ローカルメモリを宣言する |
| Hardware <n> <a> <s> | メモリマップドＩ／Ｏ領域を宣言する |
| State <f> [Clear] | 状態変数用ファイルを宣言する |
| Obey <f> | <f>に格納されたコマンドに従う |
| Quit | ＩＳＳを終了する |
| Help | このテキストを印字する |

テーブル１．　ＩＳＳへのコマンドラインインタフェース

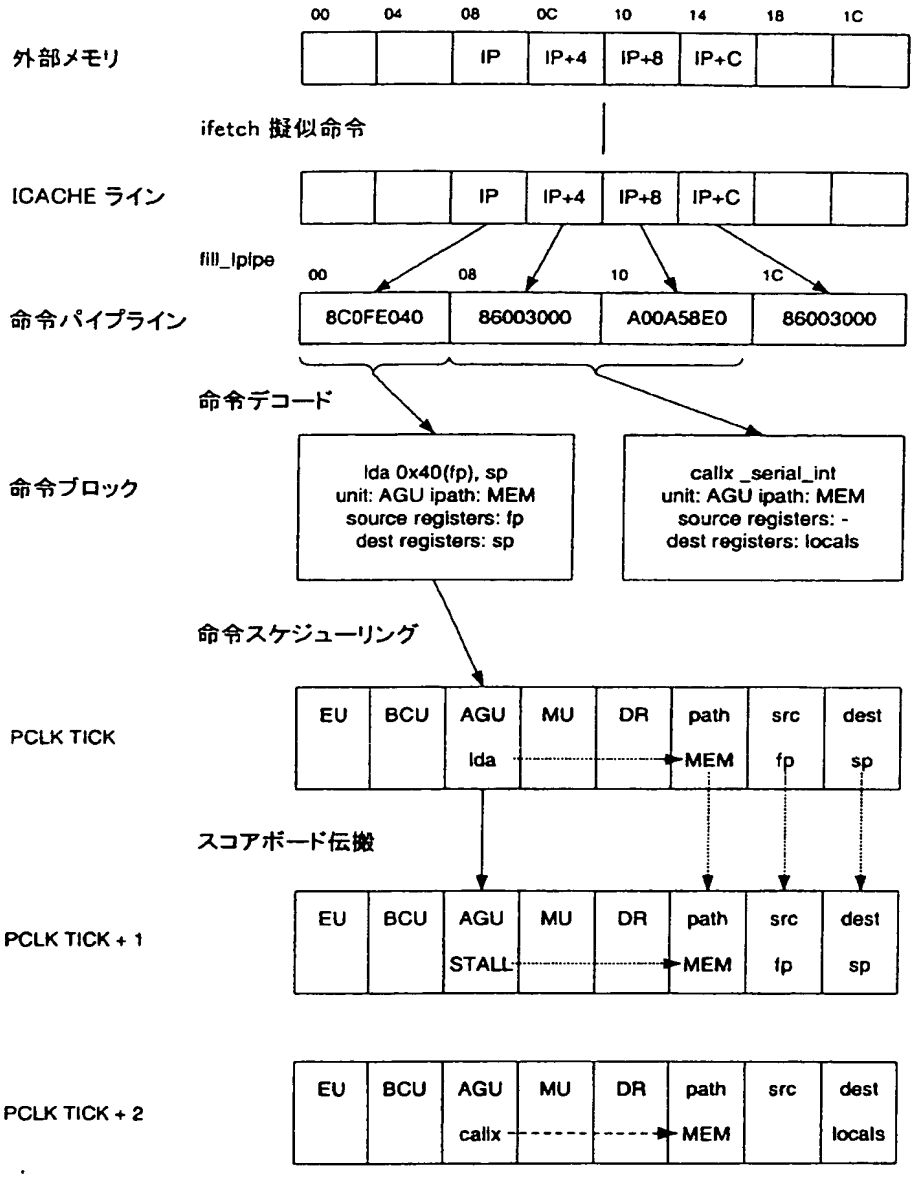
【図１３】



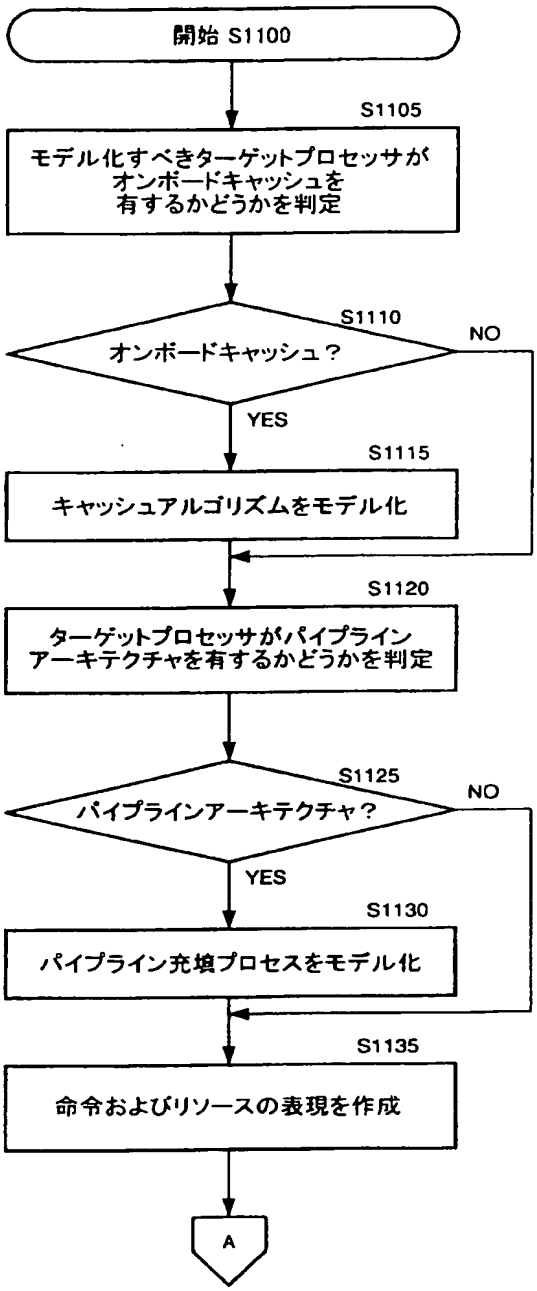

【図１６】



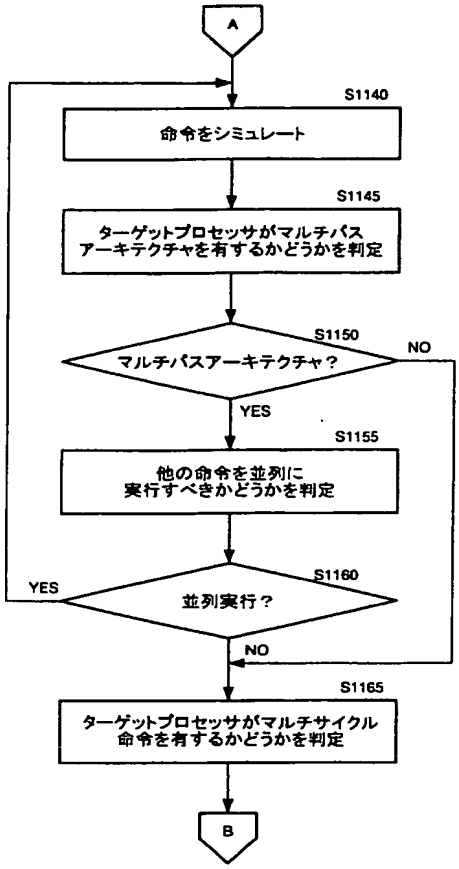
【図６】


外部メモリ
00 04 08 0C 10 14 18 1C
IP IP+4 IP+8 IP+C
ifetch 擬似命令
ICACHE ライン
IP IP+4 IP+8 IP+C
fill_Ipipe
命令パイプライン
00 08 10 1C
8C0FE040 86003000 A00A58E0 86003000
命令デコード
命令ブロック
lda 0x40(fp), sp
unit: AGU ipath: MEM
source registers: fp
dest registers: sp
callx _serial_int
unit: AGU ipath: MEM
source registers: -
dest registers: locals
命令スケジューリング
PCLK TICK
EU BCU AGU MU DR path src dest
lda MEM fp sp
スコアボード伝搬
PCLK TICK + 1
EU BCU AGU MU DR path src dest
STALL MEM fp sp
PCLK TICK + 2
EU BCU AGU MU DR path src dest
callx MEM locals

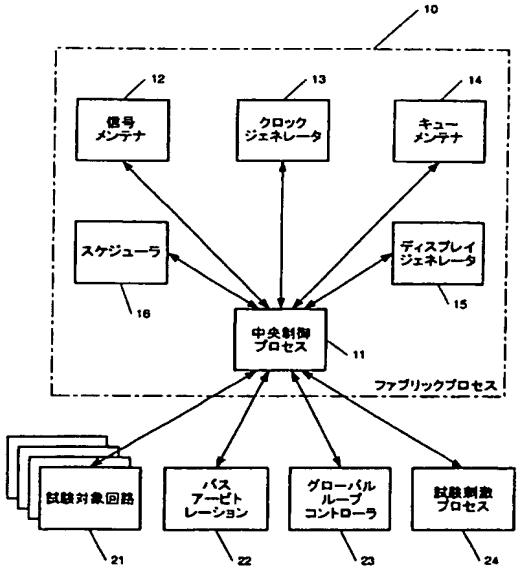
【図７】


開始 S1100
S1105
モデル化すべきターゲットプロセッサが
オンボードキャッシュを
有するかどうかを判定
S1110
オンボードキャッシュ？
NO
YES
S1115
キャッシュアルゴリズムをモデル化
S1120
ターゲットプロセッサがパイプライン
アーキテクチャを有するかどうかを判定
S1125
パイプラインアーキテクチャ？
NO
YES
S1130
パイプライン充填プロセスをモデル化
S1135
命令およびリソースの表現を作成
A

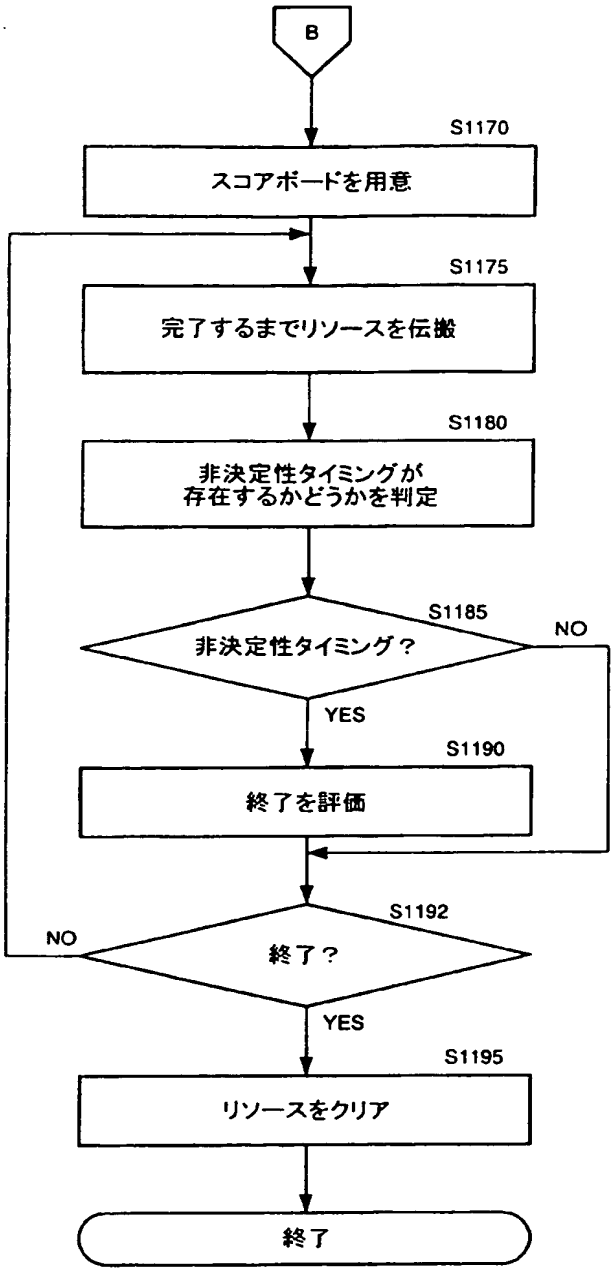
【図８】


A
S1140
命令をシミュレート
S1145
ターゲットプロセッサがマルチパス
アーキテクチャを有するかどうかを判定
S1150
マルチパスアーキテクチャ？
NO
YES
S1155
他の命令を並列に
実行すべきかどうかを判定
S1160
並列実行？
YES
NO
S1165
ターゲットプロセッサがマルチサイクル
命令を有するかどうかを判定
B

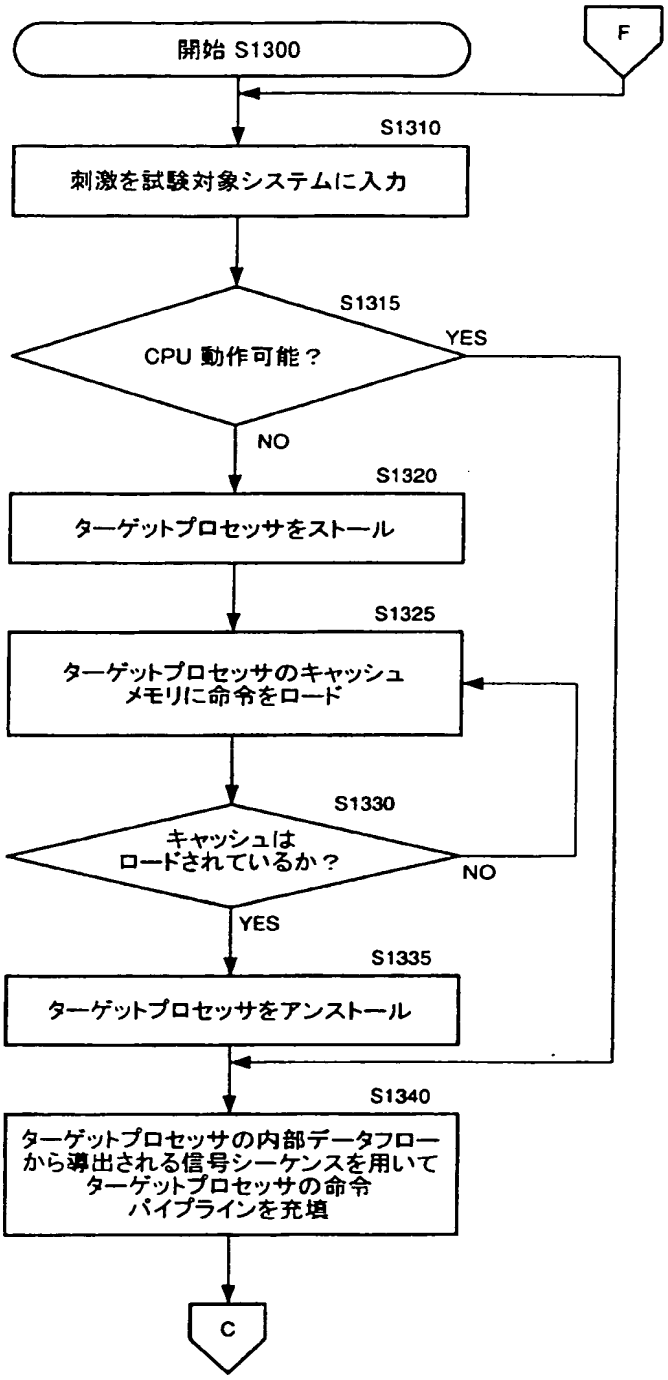
【図１４】


10
12
信号
メンテナ
13
クロック
ジェネレータ
14
キュー
メンテナ
スケジューラ
16
ディスプレイ
ジェネレータ
15
中央制御
プロセス
11
ファブリックプロセス
試験対象回路
21
バス
アービト
レーション
22
グローバル
ループ
コントローラ
23
試験刺激
プロセス
24

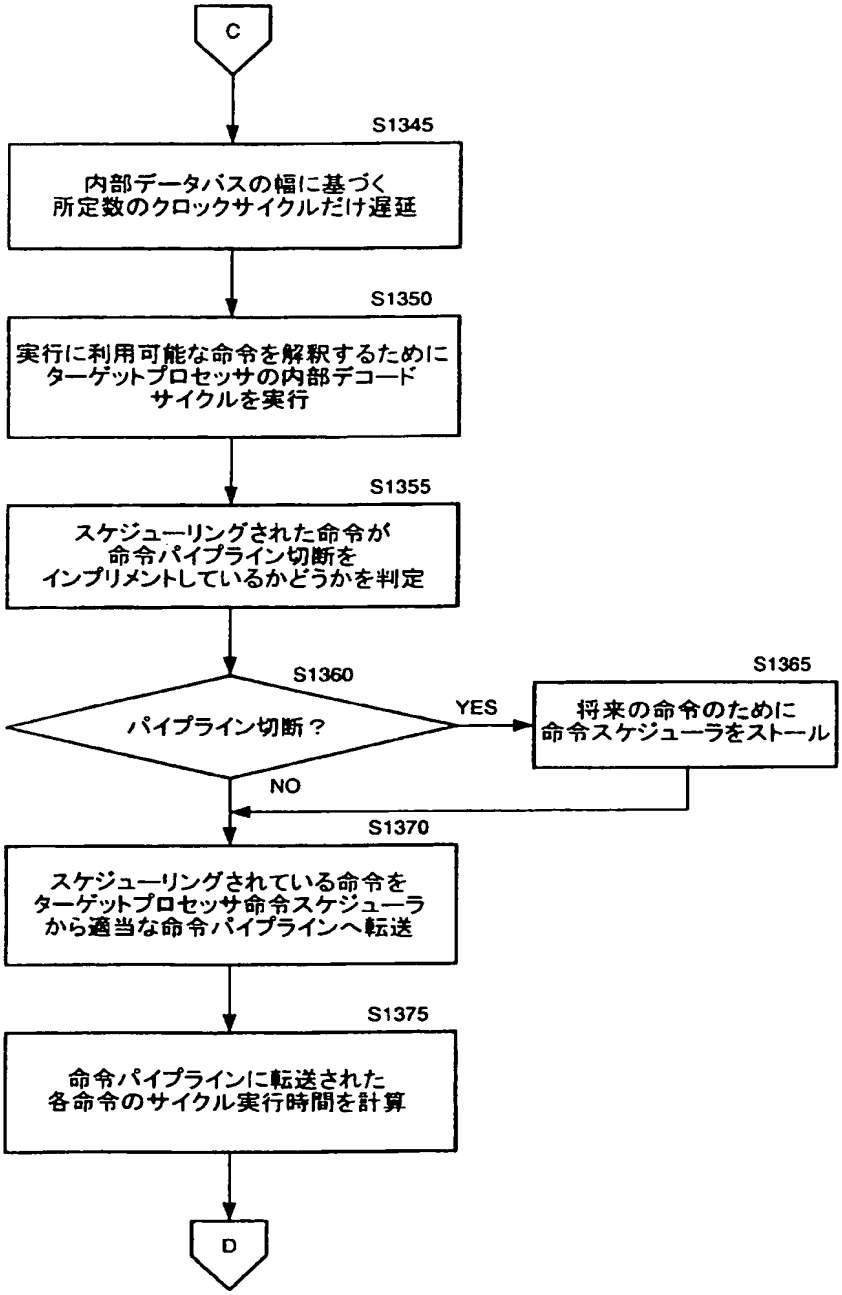
【図９】


B
S1170
スコアボードを用意
S1175
完了するまでリソースを伝搬
S1180
非決定性タイミングが
存在するかどうかを判定
S1185
非決定性タイミング？
NO
YES
S1190
終了を評価
S1192
終了？
NO
YES
S1195
リソースをクリア
終了

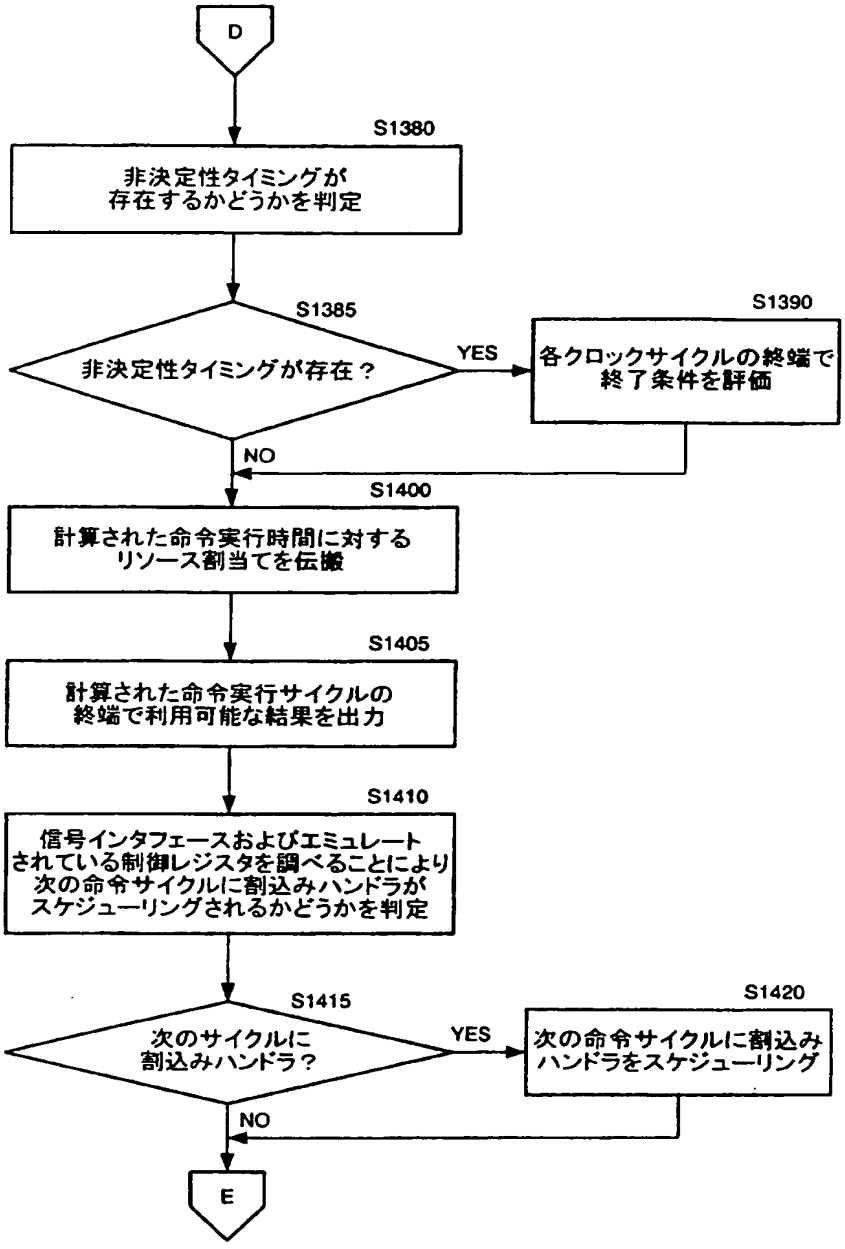
【図１０】


開始 S1300
F
S1310
刺激を試験対象システムに入力
S1315
CPU 動作可能？
YES
NO
S1320
ターゲットプロセッサをストール
S1325
ターゲットプロセッサのキャッシュ
メモリに命令をロード
S1330
キャッシュは
ロードされているか？
NO
YES
S1335
ターゲットプロセッサをアンストール
S1340
ターゲットプロセッサの内部データフロー
から導出される信号シーケンスを用いて
ターゲットプロセッサの命令
パイプラインを充填
C

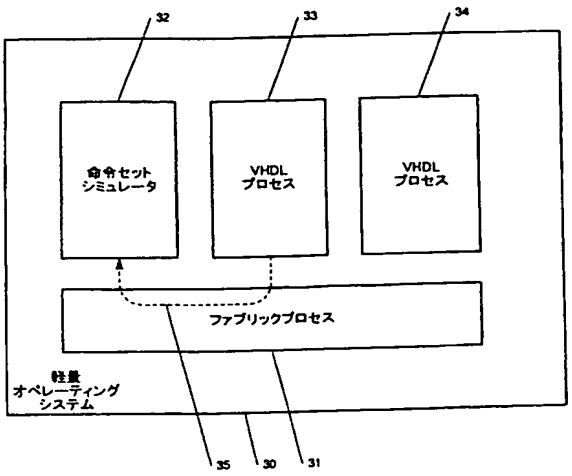
【図１１】


C
S1345
内部データバスの幅に基づく
所定数のクロックサイクルだけ遅延
S1350
実行に利用可能な命令を解釈するために
ターゲットプロセッサの内部デコード
サイクルを実行
S1355
スケジューリングされた命令が
命令パイプライン切断を
インプリメントしているかどうかを判定
S1360
パイプライン切断？
YES
S1365
将来の命令のために
命令スケジューラをストール
NO
S1370
スケジューリングされている命令を
ターゲットプロセッサ命令スケジューラ
から適当な命令パイプラインへ転送
S1375
命令パイプラインに転送された
各命令のサイクル実行時間を計算
D

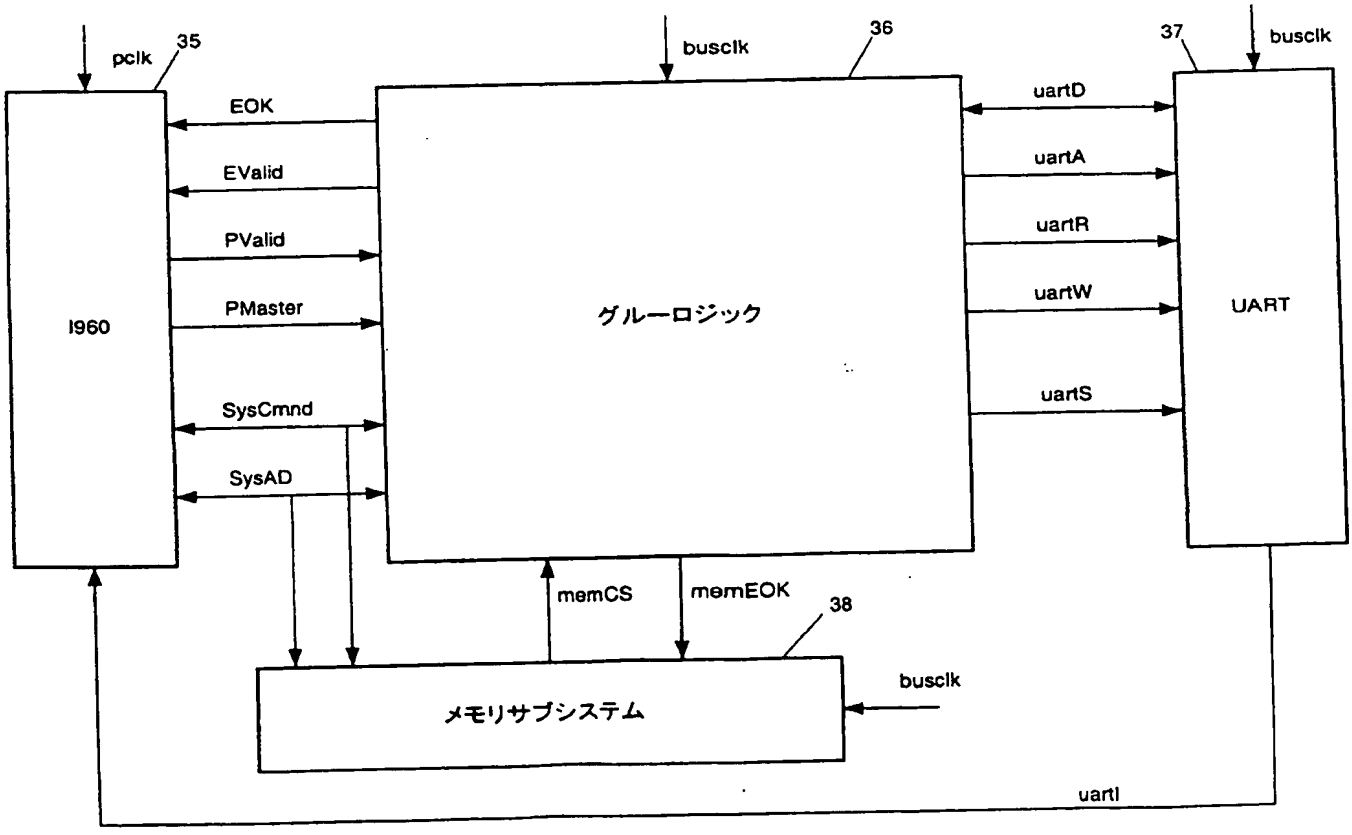
【図１２】


D
S1380
非決定性タイミングが
存在するかどうかを判定
S1385
非決定性タイミングが存在？
YES
S1390
各クロックサイクルの終端で
終了条件を評価
NO
S1400
計算された命令実行時間に対する
リソース割当てを伝搬
S1405
計算された命令実行サイクルの
終端で利用可能な結果を出力
S1410
信号インタフェースおよびエミュレート
されている制御レジスタを調べることにより
次の命令サイクルに割込みハンドラが
スケジューリングされるかどうかを判定
S1415
次のサイクルに
割込みハンドラ？
YES
S1420
次の命令サイクルに割込み
ハンドラをスケジューリング
NO
E

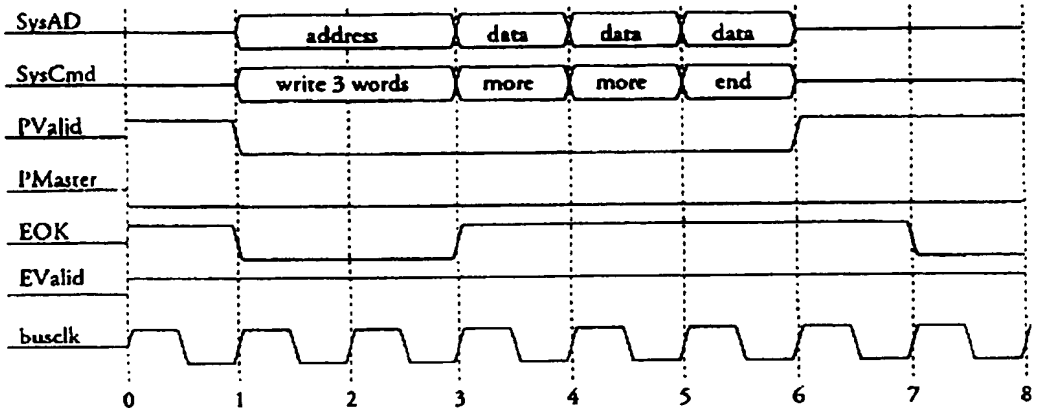
【図１５】


32
33
34
命令セット
シミュレータ
VHDL
プロセス
VHDL
プロセス
ファブリックプロセス
性能
オペレーティング
システム
35
30
31

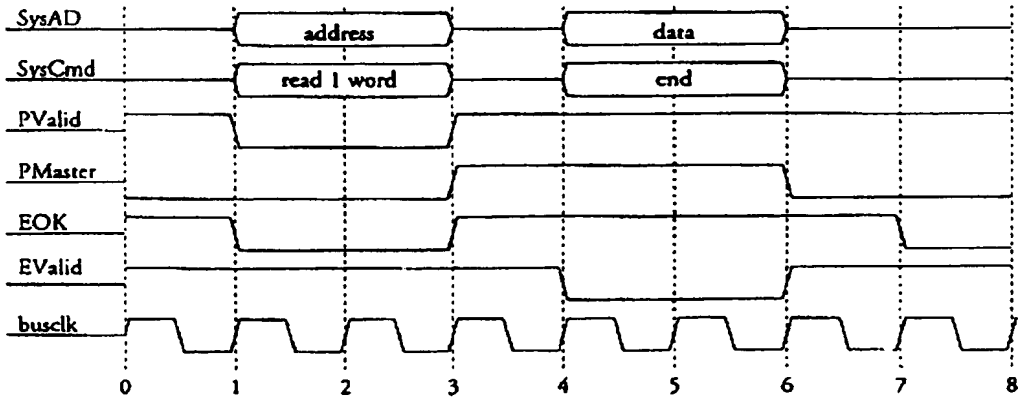
【図１７】


pclk
35
busclk
36
37
busclk
EOK
EValid
PValid
PMaster
SysCmnd
SysAD
I960
グルーロジック
uartD
uartA
uartR
uartW
uartS
UART
memCS
memEOK
38
busclk
メモリサブシステム
uartI

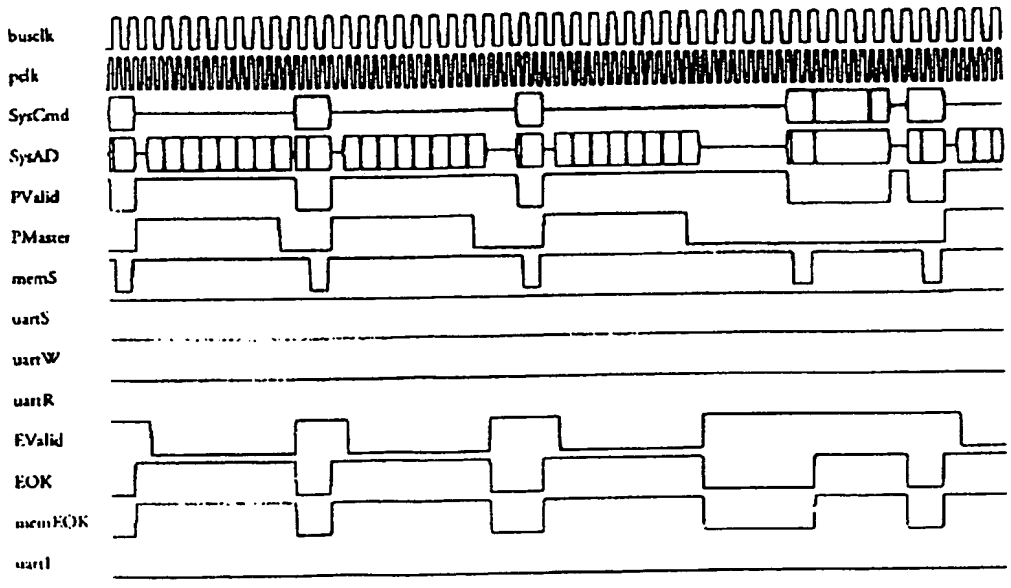
【図１８】


SysAD
SysCmd
PValid
PMaster
EOK
EValid
busclk
address
data
data
data
write 3 words
more
more
end
0
1
2
3
4
5
6
7
8

【図１９】


SysAD
SysCmd
PValid
PMaster
EOK
EValid
busclk
address
data
read 1 word
end
0
1
2
3
4
5
6
7
8

【図２０】


busclk
pclk
SysCmd
SysAD
PValid
PMaster
memS
uartS
uartW
uartR
EValid
EOK
memEOK
uartI